AN EGRESS QUEUE MANAGER
FOR 8 X 100 MBPS AND 1 X 1GBPS
ETHERNET SWITCH PORT CONTROLLERS

by

BARRY GORDON MOSS

B.A.Sc., Simon Fraser University, 2002

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

IN THE SCHOOL OF ENGINEERING SCIENCE

SIMON FRASER UNIVERSITY

Decemeber 2002

Approval

Name: Barry Gordon Moss

Degree: Master of Engineering

Title of thesis: An Egress Queue Manager for 8 x 100 Mbps and 1 x 1 Gbps Ethernet Switch Port Controllers

Presentation Committee:

Chair:

Dr. Paul Ho
Project Committee Chair

Dr. Ljiljana Trajkovic
Senior Supervisor
School of Engineering Science

Dr. Steven Hardy
Supervisor
School of Engineering Science

Steve Dabecki
External Supervisor
PMC-Sierra, Inc

Date approved:

Abstract

The project consists of the design of an Egress Queue Manager (EQM), a major functional block used in PMC-Sierra's PM3370 (8 x 100 Mbps) and PM3380 (1 x 1 Gbps) Ethernet Switch Port Controller integrated circuits. The EQM acts as the egress port controller in an output associated input buffered switch, providing the option of three different traffic scheduling algorithms, including variants of weighted fair queuing. While designed to operate within the environment of an Ethernet packet switch, the design is flexible enough to be adapted to other packet switched environments (such as a IP based switch) with minimal modifications.

The project consisted of the redefinition, design, verification and documentation of the EQM block, comprising approximately 70,000 gates of logic. The design was captured at the register transfer level using the Verilog hardware description language and then synthesized into a gate-level netlist. The project also includes a suite of simulations designed to verify the functionality and performance of the design, and static timing analysis to guarantee that the synthesized design was capable of operating at a clock rate of 83 MHz.

Dedication

This project is dedicated to my family and friends who have encouraged me through the long process of pursuing this degree and to my former coworkers at PMC-Sierra who made this project possible. Thank you.

Acknowledgments

Many thanks to the following individuals for their contributions to the EQM project:

Steve Dabecki at PMC-Sierra who was the technical lead in charge of the SWIT sub-section of the PM3370 and PM3380 port controller chips.

Joshua Salvador, who as an engineering co-op executed the performance simulations for the EQM design and who helped complete the second simulation test bench.

Tom Alexander from PMC-Sierra's Portland office, who invented (and re-invented) the EXACT protocol, architected the EXACT chip set, and drafted the initial EQM specification.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The Egress Queue Manager (EQM) is a major functional block used in PMC-Sierra's PM3370 [1] (8 x 100 Mbps) and PM3380 [2] (1 x 1 Gbps) Ethernet Switch Port Controller integrated circuits. Along with PMC-Sierra's PM3390 Switch Fabric Device, these ICs form the EXACT flexible and stackable Ethernet switching architecture, capable of maintaining full line rate switching on all ports even when servicing minimum sized (64 byte) packets. The EQM operates as an 8-port single stream output scheduler controller when used in the PM3370, and operates as a single port multi-stream scheduler when used in the PM3380.

Packets are transmitted across the switch fabric in variable sized data blocks with a maximum size of 240 bytes. The EQM manages the handshaking protocol for service requests, collects transmitted data statistics and forwards the data blocks a downstream buffer for frame reassembly.

The EQM arbitrates between service requests submitted by ingress queue managers (IQM), which have traffic queued for the particular egress ports assigned to the EQM. Within a given class of service (4 are supported in the EQM), the EQM uses a simple round robin protocol. However, the EQM implements three parameterizable queuing or scheduling algorithms, which determine how bandwidth is allocated between the service classes. The simplest algorithm supported is a strict priority scheduler in which the highest service class with pending requests is always processed first. Strict priority scheduling can be used to implement the IEEE 802.1p standard [3]. The EQM also supports a weighted fair scheduling method which achieves many of the goals of the weighted fair queuing algorithm first described by Demers, Keshav and Shenkeri [4] but using a method much easier to implement in silicon. Finally, a hybrid

method, which combines features of both priority and weighted queuing, is implemented.

Within a service class, the EQM supports fair queuing (round robin) service of the individual source queues.

2 The System Environment

2.1 Port Controller Overview

The EQM acts as the egress controller in an output associated, input buffered switch. In this architecture, packet data is stored by an ingress queue manager (IQM) in queues corresponding to individual output ports. The ingress controller buffers a complete packet before sending a queue allocate (QA) message through the switching fabric to the EQM, where the data is destined. The switching fabric consists of one or more linked rings implementing PMC-Sierra's EXACT protocol.

The EQM does not directly access frame data as this is buffered at the ingress port, which may be on the same or a different port controller. Frame data is stored in memory as partial packet buffers (PPB's). Depending on the system configuration, each PPB contains up to 120 or 240 bytes of frame data together with header and trailer information. Independent of the size of the payload, each data block is packed into a 256 byte Partial Packet Buffer. (The 240 byte option made most efficient use of the PPB memory space; however, the 120 byte option allowed for finer granularity within the fabric). PPB's are chained together to form a linked list. The data contained in a single PPB is transmitted across the switch fabric.

The structure of the PPB's is illustrated in Figure 1.

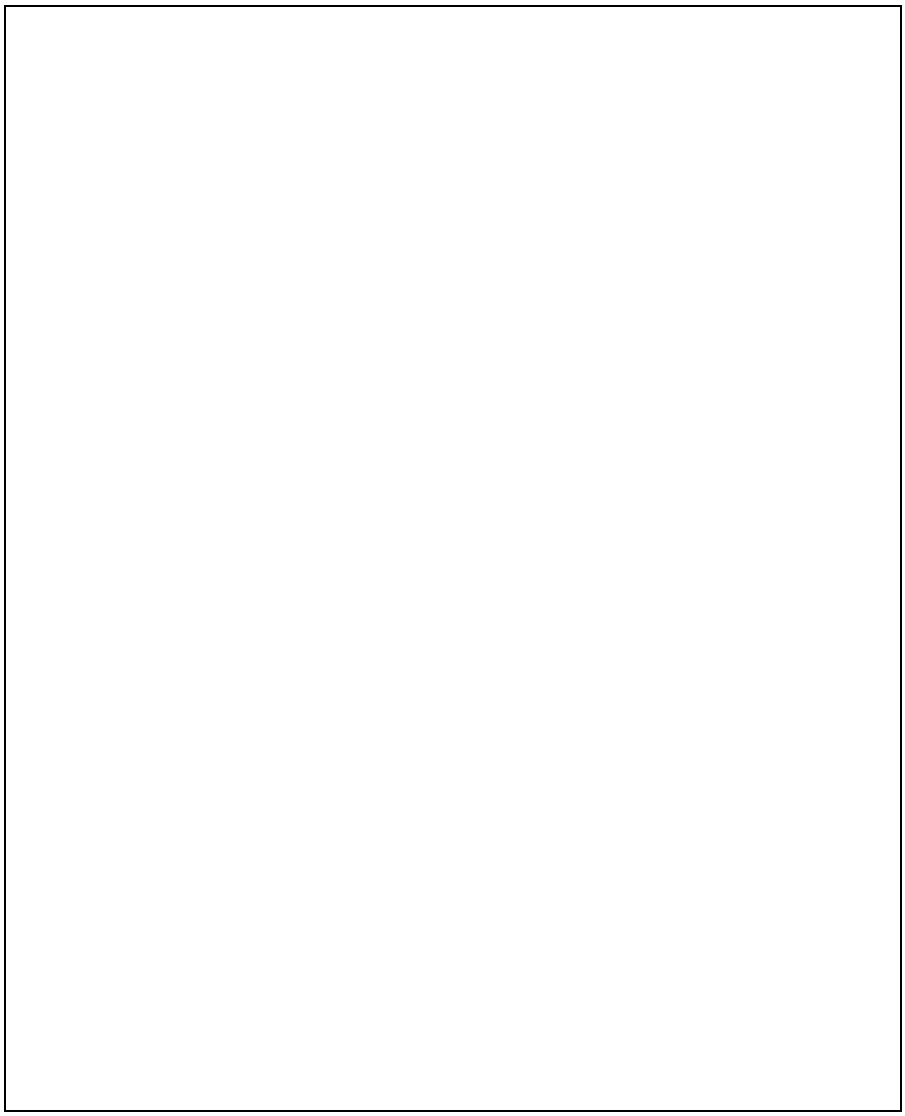

Figure 1. Partial Packet Buffer Chaining Structure. Ethernet frames are segmented and placed into queues based on the destination port and priority level. Each queue contains pointers to the first Partial Packet Buffer (PPB) of the linked list of PPBs holding a given frame [5].

While the EQM is not protocol specific, it was implemented within an Ethernet switching system. A maximum length Ethernet frame (1522 bytes with optional VLAN tag) will require up to 14 chained PPB's to hold the complete frame, depending on the data block (DB) payload size, which is configured externally to the EQM.

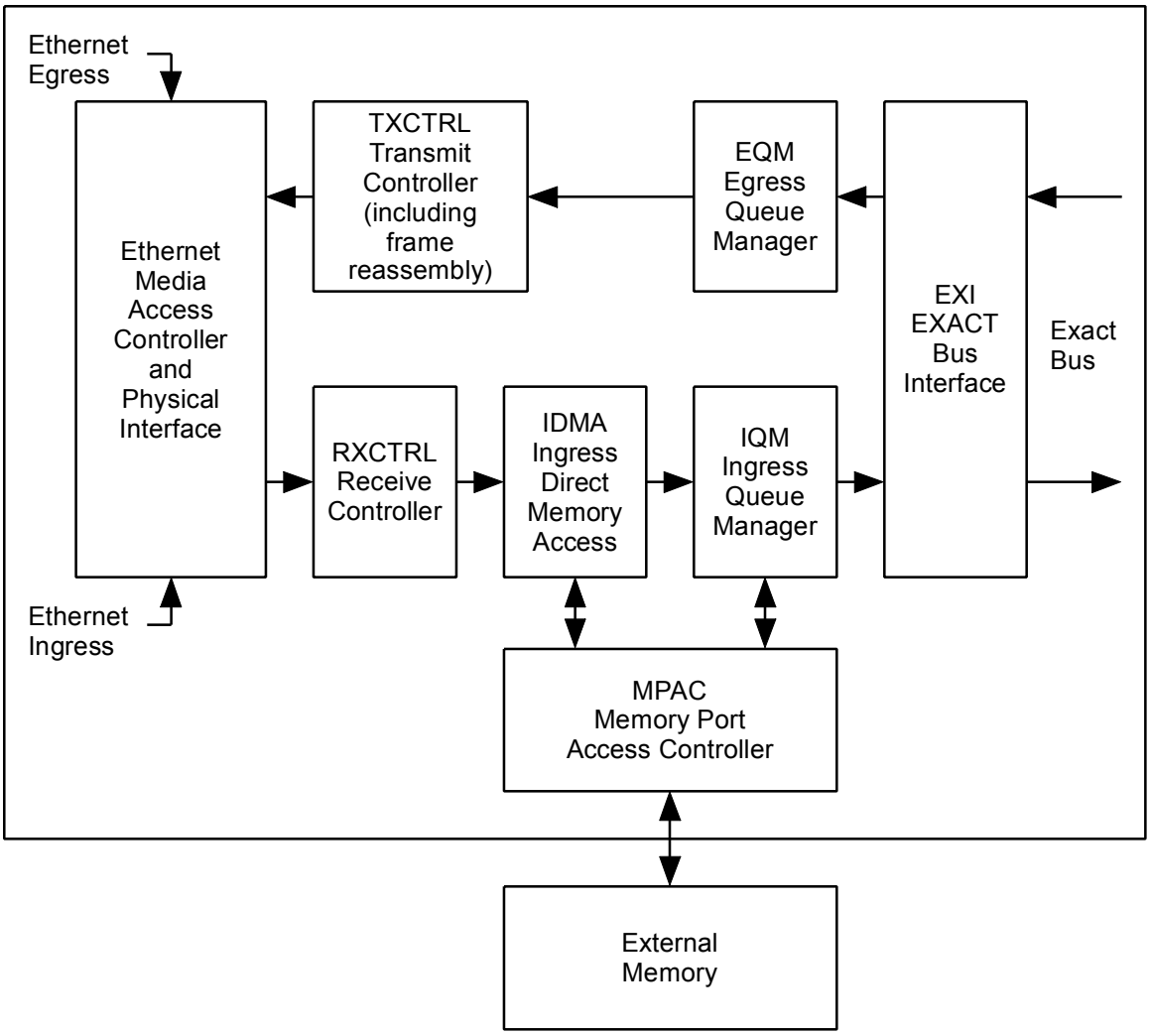

Figure 2. Simplified Port Controller Block Diagram.

Data is queued on the ingress side of the device. Control and Data messages are sent over the EXACT bus to an egress controller on the addressed port controller. Figure 2 shows a simplified block diagram or the port controller. Received Ethernet packets pass are processed by the Ethernet MAC and Receive Controller. The Ingress DMA segments the packet payloads into 240 segments and stores them in an external memory device as a chained PPB structure (see Figure 1). When a complete packet payload has been segmented into a chain of PPBs, the IDMA passes the head pointer of the chain to the

Ingress Queue Manager. The IQM places the pointer in a queue corresponding to the output port and priority (or class of service). The IQM communicates with the EQM corresponding to the egress port using the EXACT protocol (see section 2.2 following). Packet data fragments are passed through to the EQM to the Transmit Controller (TXCTRL) where the fragments are reassembled in buffers. When a complete packet has been reassembled in the TXCTRL, it is transmitted through the MAC and physical interface on the appropriate Ethernet egress port.

The EXI serves as the physical interface to the EXACT ring for the IQM and EQM, handling clock rate conversions and data path width conversions with shallow FIFOs.

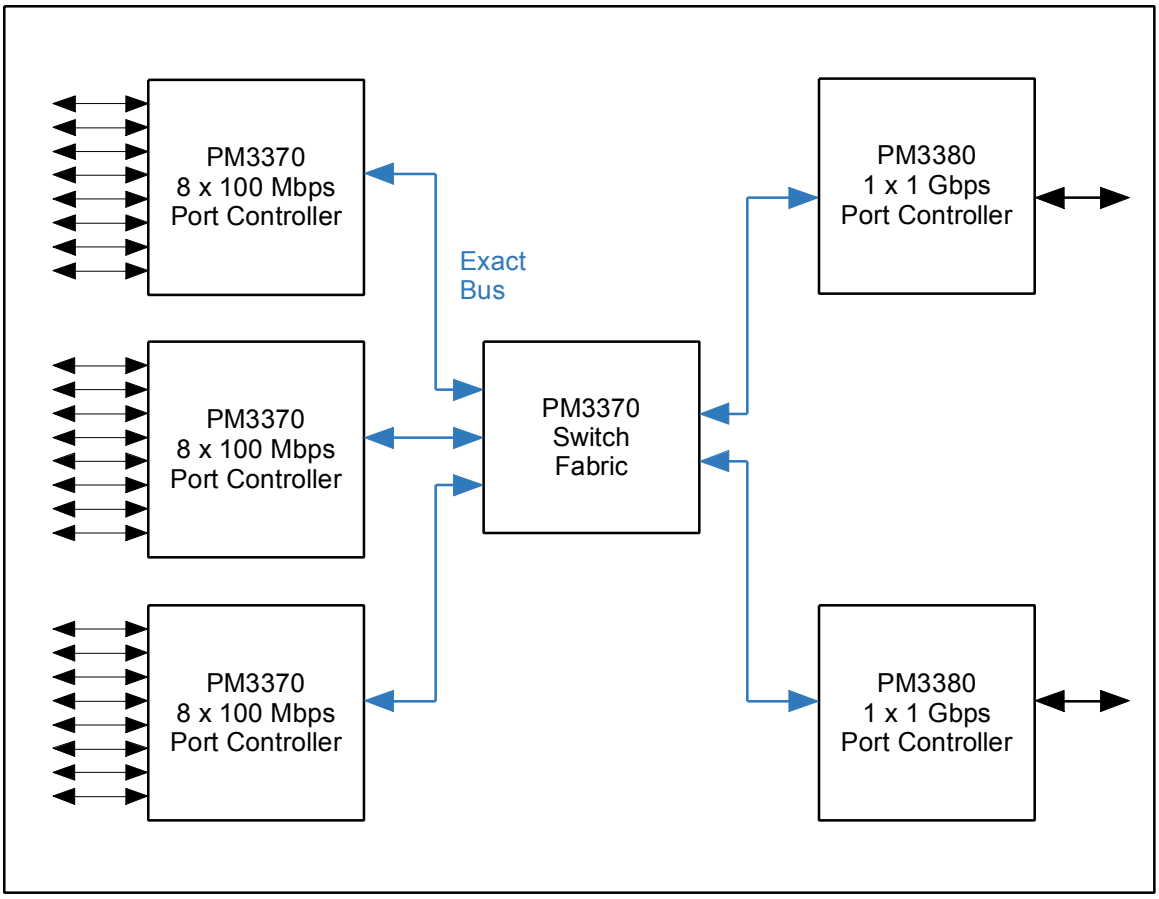

Figure 3. An EXACT Based Ethernet Switch. A 24+2 Ethernet Switch created using EXACT port controllers and the ENCORE switch fabric devices. Smaller switches (e.g. an 8+1 or 16+0 switch) can be created without the switch fabric IC by connecting the EXACT Bus in a ring.

2.2 The EXACT protocol

The EXACT protocol is used to transfer data across the switching fabric, which may be a simple ring connection or a switch fabric IC.

When a given IQM has queued a packet for an output port, it will send a QA message across the fabric to notify the corresponding EQM that a complete packet is available for a given output port (see Figure 4).

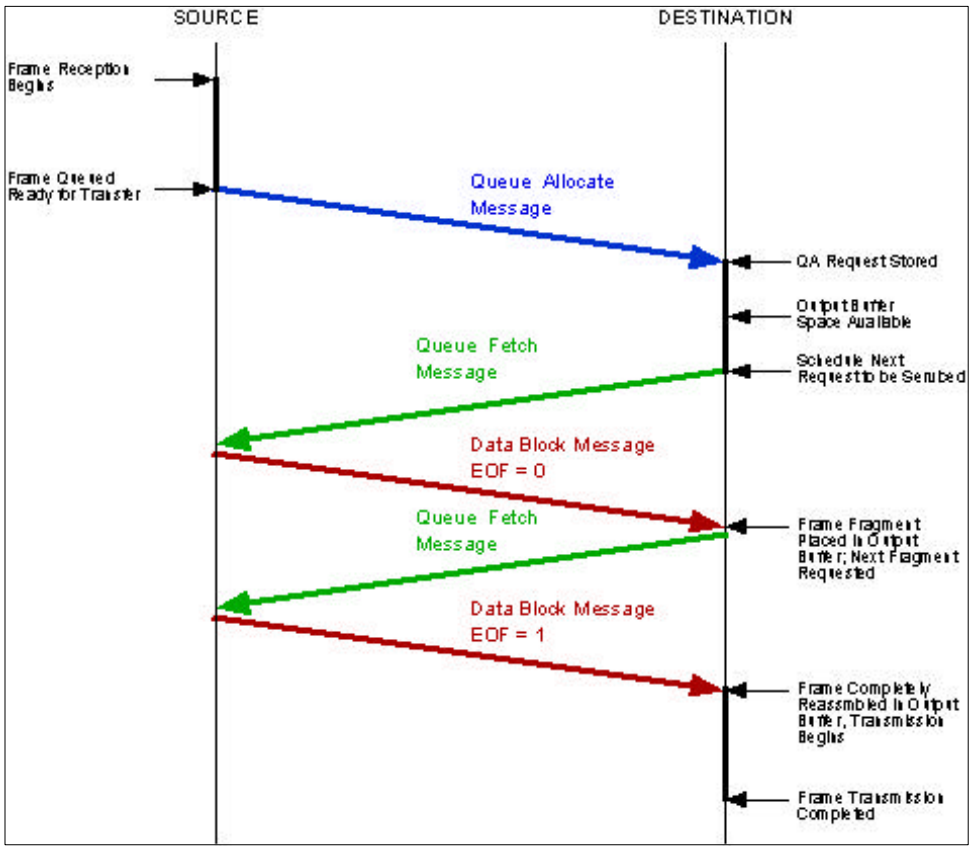

Figure 4. EXACT Protocol Flow Diagram. The source device (usually an IQM) signals available traffic by sending a QA message to the destination deivce (usually an EQM). When the destination device is ready to service the QA request, it returns a QF message, allowing the source to send a single DB message. The QF/DB exchange continues until the entire packet has been transmitted across the EXACT ring or switch fabric.

When the EQM is ready to accept that packet, it will send a Queue Fetch (QF) message, which allows the IQM to send a single data block (DB) message containing the first fragment of a packet, as contained in the payload of a single PPB. If there is more data in the packet, the DB message will have the EOF flag cleared in the message header. If the DB contains the end of the packet/frame, the EOF is set and the IQM may immediately send another QA message if there is one or more additional packets queued for the given output port. When the EQM receives a DB message it checks the EOF flag. If the flag is clear, the EQM

will return another QF message when it is ready to accept another DB message. The message formats are described in greater detail later in this report.

System simulations done by the project architect showed that while this simple version of the protocol was sufficient to sustain line rates for 100 Mbps ports, the latency across the switch fabric was too great for 1 Gbps ports. Consequently, the protocol was modified for output ports operating at the higher rate.

The modified protocol allowed the IQMs to send up to 15 QA messages which would be stored by the EQMs. In return the EQM is allowed to issue up to one QF message for each QA message stored up to a maximum of 10. This allows the EWM to accept BD messages for up to 10 packets in parallel while in 1 Gbps mode. These packets could all originate from the same IQM or from multiple IQMs.

3 Functional Description

3.1 Overview

Figure 5 shows the block diagram of the EQM, which consists of the following modules:

- Queue Allocate/Data Block Processor (QADBP).
- Transmit Pending Arrays (TPAs).
- Queue Fetch Scheduler (QFS).
- Transmit Frame Reassembly Buffer Interface (TFRBI).
- Transmit Statistics Transaction FIFO (TSTF).
- Statistics Updater and Memory Port Access Controller (MPAC) SRAM Interface (SUMI).
- Control Registers and Control Register Interface (CRI) Logic.
- Built-in Self Test (BIST) Sequencer for RAM modules.
- Test Registers and Interface Logic.

Note: The test registers and BIST circuits have been omitted from the block diagram.

A significant design goal was to reuse as much logic and in particular the TPA memories in the two operating modes (single port 1 Gbps and eight port 100 Mbps). Whenever possible, the sub-modules were designed to support both modes. This approach was successful for all blocks except the QFS which required very different designs for the two modes. Thus the QFS essentially contains two designs and the appropriate logic is enabled based on the mode selected.

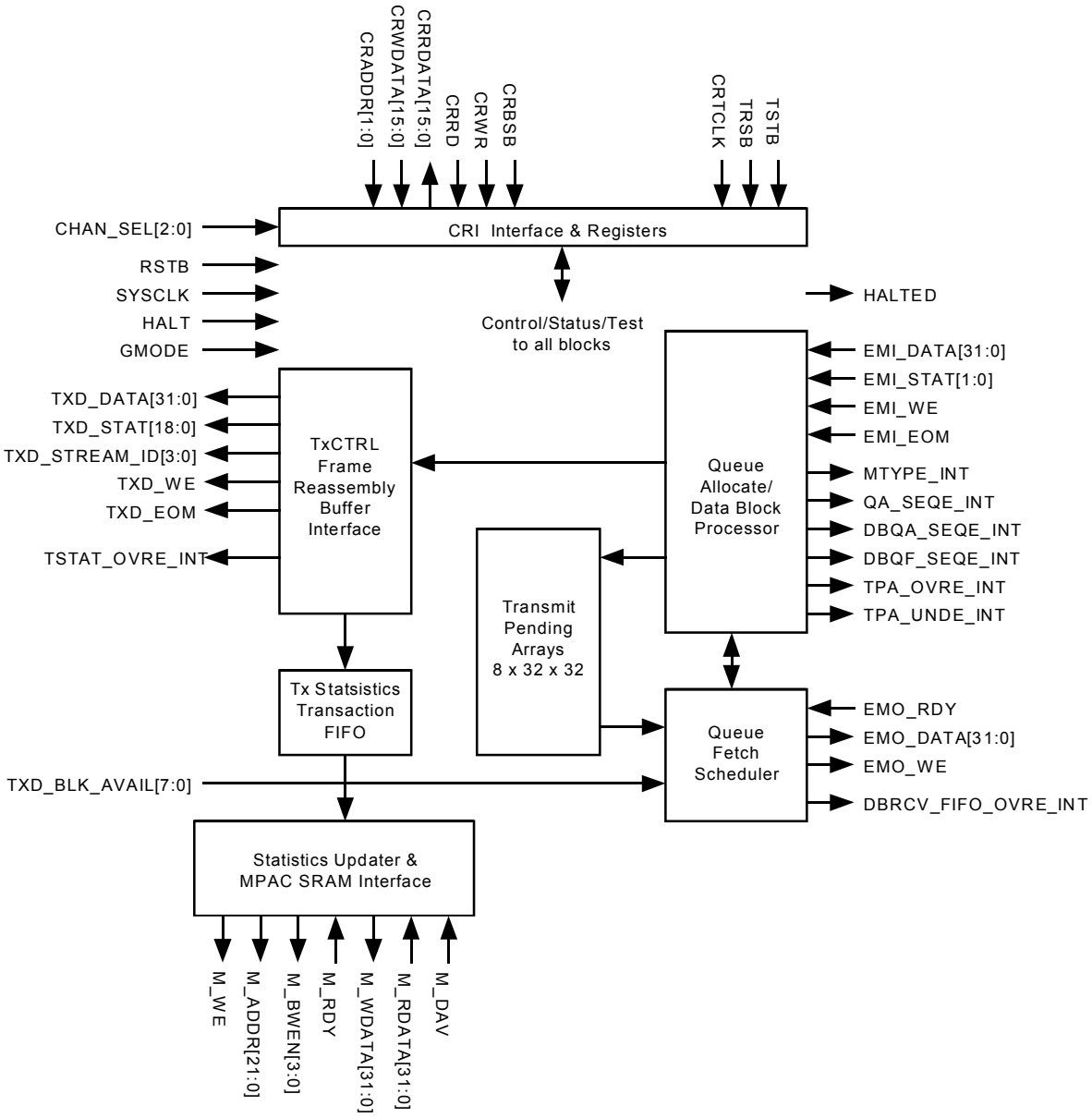

Figure 5. EQM Block Diagram. Test registers and memory BIST are not shown.

3.2 Queue Allocate/Data Block Processor (QADBP)

The Queue Allocate / Data Message Processor is responsible for the following:

- Receiving of queue allocate (QA) and data block (DB) messages.
- Parsing of the TYPE field within the message header to determine message type.
- Parsing of the other header fields to determine channel destination, source port, and priority/class of service.
- 100 Mbps mode: Setting the appropriate bit in the corresponding Transmit Pending Array (TPA) upon receipt of a QA message.
- 1 Gbps mode: Incrementing the appropriate 4-bit count in the Transmit Pending Array (TPA) upon receipt of a QA message.
- 100 Mbps mode: Upon receipt of a DB message, routing the payload to the TxCTRL Frame Reassembly Buffer interface (TFRBI) and setting the DB received (DBRCV) flag for the appropriate destination channel. The DBRCV flag causes the QFS to send the next QF message as soon as space is available in the corresponding downstream Transmit Frame Reassembly Buffer. If the DB header has the End of Frame (EOF) bit set, then the appropriate bit in the Transmit Pending Array is cleared, and the EOF flag for the appropriate destination channel is set. The EOF flag causes the QFS to resume the scanning the TPA for that channel according to the scheduling algorithm.
- 1 Gbps mode: Upon receipt of a DB message, routing the payload to the TxCTRL Frame Reassembly Buffer interface (TFRBI) and placing an entry in the Queue Fetch Scheduler's (QFS) DB received FIFO (DBRF) indicating the

SPID, P and EOF fields of the DB, which causes the QFS to send the next QF message (unless the EOF flag is set) as soon as space is available in the TFRBI. If the DB header has the End of Frame (EOF) bit set, then the appropriate count in the Transmit Pending Array is decremented.

The QADBP receives messages from the upstream EXI. These messages will be of two types. These will be either a queue allocate message (QA), which indicates that a source has data for one of the channels, or they will be data blocks (DB's), which contain partial data blocks constituting the complete frame. The QADBP is capable of processing a mix of QA and DB messages with minimal latency; however, the EXI must guarantee a minimum of two inactive cycles on the EMI interface between subsequent messages since the QADBP requires a minimum of 3 SYSCLKs to read-modify-write the TPA RAM.

The format of queue allocate and data block messages as presented by the EXI are illustrated in Figure 6 and Figure 7 respectively. Note that the EQM only supports 256 SPIDs and 8 DCIDs. The unused most significant bits in each field (SPID[9:8] and DCID[5:3]) will always be ignored.

Figure 6. Queue Allocate (QA) Message Format. See Table 1 for field definitions. QA messages are used to signal that an input queue has a complete packet ready to transmit to an output port associated with this EQM.

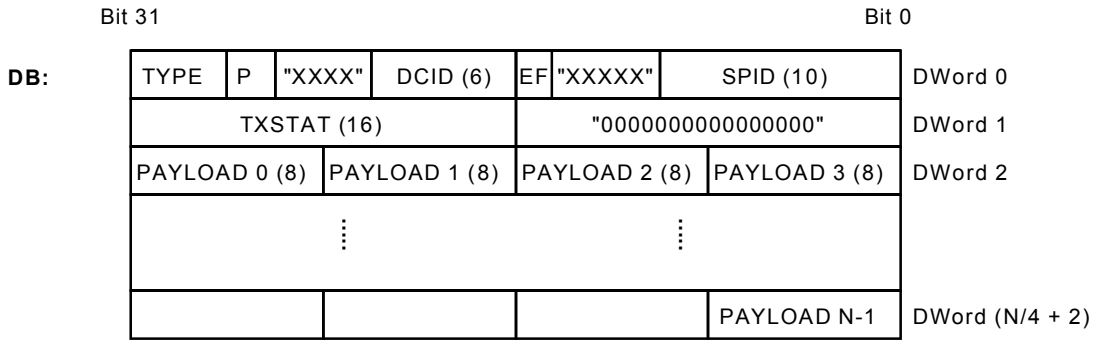

Figure 7. Data Block (DB) Message Format. DB messages contain packet data from a single PPB. If this is the last block of data in a given frame, the EF (end-of-frame) bit will be set. See Table 1 for field definitions.

Table 1. QA and QB Message Field Descriptions.

Field	Bits	Description
TYPE	4	Message Type. Defines up to 16 possible message types, of which the EQM requires to know only 3: QA, QF, DB. 0000 - Data Block (DB) 0010 - Queue Allocate (QA) 0100 - Queue Fetch (QF) Other codepoints are reserved.
P	2	Priority of the frame. Every frame is allocated one of four priority levels. The IDMA passes this field on to the queue manager. The queue manager will queue the frame according to the following: '00' - priority 0 '01' - priority 1 '10' - priority 2 '11' - priority 3 Note that priority 0 is the highest priority and priority 3 is the lowest.
EF	1	End of Frame (EOF). When EOF is set, this indicates the last data block (PPB) of the frame.
SPID	10	Source Port Identifier This field indicates the EXACT ring source port from which the queue allocate was received. This field is returned as the destination port in a QF message. Note there is not necessarily a 1:1 correspondence between Ethernet ports and EXACT ring source ports. An EXACT port controller may queue frames from several Ethernet ports into a single queue which will have a single EXACT ring ID.
DCID	6	Destination Channel Identifier. This field indicates the destination channel for which the queue allocate or data is intended.
TXSTAT	16	Transmit Statistics. A 16-bit field which is used to index into the statistics memory.
PAYLOAD	8+	Data Payload. This variable length field contains the packet data to be passed to the TxCTRL block. The field may vary from 1 to 240 bytes in length. The EQM does not examine or process the payload field.
other bits	-	For messages that are received by the EQM (QA and DB), bits marked as "X" are ignored. For messages that are transmitted by the EQM (QF), these fields are set to 0. Note that some of these fields are overwritten by the upstream EXI block, to indicate a pseudo source address.

The QADBP checks the header fields of the messages to first determine the type of message, and then the remaining fields to determine the subsequent processing.

3.2.1 Queue Allocate Processing

A QA message indicates to the EQM that a source has data buffered for one of the channels supported by the EQM. On receiving a QA in 100 Mbps mode, the QADBP reads the appropriate line from the destination channel's TPA, sets the appropriate bit for the source port and priority/class of service, and then writes the modified line back to the TPA. In 1 Gbps mode, the QADBP reads the appropriate 4 lines from the TPAs to assemble a 4-bit count value, increments this value and then writes the modified lines back to the TPA.

In 100 Mbps mode, it is an error for the QADBP to receive a QA for a DCID/SPID/priority combination for which the QA bit is already set. In this case, the QADBP sets the QA_SEQE_INT interrupt flag, and places the source and destination port numbers in the appropriate error status registers.

In 1 Gbps mode, it is an error for the QADBP to receive a QA for a SPID/P combination for which TPA count has reached its maximum value of 15. In this case, the QADBP sets the TPA_OVRE_INT interrupt flag, and places the source and destination port numbers in the appropriate error status registers. If another TPA Overflow or sequence error occurs before the error status registers are read, the error status registers will be overwritten with the new information.

3.2.2 Data Block Processing

A DB message is sent to the EQM as a result of the source port receiving a queue fetch from the EQM. On receiving a DB, the QADBP routes the data block to the TxCTRL Frame Reassembly Buffer Interface (TFRBI), which in turn writes

the data to the downstream TxCTRL Frame Reassembly Buffer (FRB). Only the payload data is transmitted to the FRB; the DB message header is removed.

The QADBP monitors the EF (end of frame) field in the data block message which indicates that this DB is the last for the frame. If this DB is not the last for the frame, then the QADBP in 100 Mbps mode immediately sets the DBRCV bit for the destination channel. If the EOF bit is set, then the QADBP clears the appropriate bit in the TPA. In 1 Gbps mode, the QADBP places an entry (including the SPID/P and EF fields) in the DBRCV FIFO of the Queue Fetch Scheduler (QFS). If the EOF bit is set, the QADBP will also decrement the corresponding count value in the TPA.

In 100 Mbps mode, it is an error for the QADBP to receive a DB for a channel which doesn't have the appropriate bit already set in its TPA, thus indicating that the DB was received without a preceding QA message or that the EOF field was set in a DB prior to the last DB for the frame. In this case, the QADBP sets the DBQA_SEQE status flag, and places the source and destination port numbers in the appropriate error status registers.

In either mode, it is also an error for the QADBP to receive a DB for which a corresponding QF message has not been sent by the QFS. In 100 Mbps mode, the QADBP checks for this condition by comparing the SPID and P output fields of the TPA_SCANNER for the appropriate channel. In 1 Gbps mode, the QADBP checks for this condition by comparing the DB header information against the SPID and P fields for all active streams from the QFS stream table If this error occurs, the QADBP sets the DBQF_SEQE status flag, and places the source and destination port numbers in the appropriate error status registers.

In 1 Gbps mode, it is an error for the QADBP to receive a DB for a SPID/P combination for which the TPA has a count of 0, thus indicating that the DB was received without a preceding QA message or that the EOF field was set in a DB

prior to the last DB for the frame. In this case, the QADBP sets the DBQA_SEQE status flag, and places the source and destination port numbers in the appropriate error status registers. If the EOF bit was set in the DB header, then the QADBP sets the TPA_UNDE status flag, and places the source and destination port numbers in the appropriate error status registers.

3.2.3 Sequence and TPA Error Handling

Sequence errors (QA, DBQA, DBQF) and TPA Overflow Underflow errors are serious protocol errors, which should occur very infrequently if ever. A single register reports the source and destination fields of the message which caused the errors is used for all error types. If another sequence error occurs before the error status registers are read, the error status registers will be overwritten with the information from the message causing the latest error. While these protocol errors were rarely if ever a problem with the production chip set, these diagnostic features proved to be invaluable while debugging the prototypes.

3.3 Transmit Pending Arrays (TPA)

3.3.1 100 Mbps mode

In 100 Mbps mode, there is a Transmit Pending Array (TPA) for each of the eight supported channels. Each TPA consists of four vectors, one vector for each class of service. (There is an independent TPA for each channel because all the TPA for each channel must be searched in parallel in order to meet the QA received to QF issued latency requirements). There is a single bit in the vector for each source port. Since there are a total of 256 supported source ports per vector and 4 classes of service vectors, each TPA consists of 1024 bits. (See Figure 8).

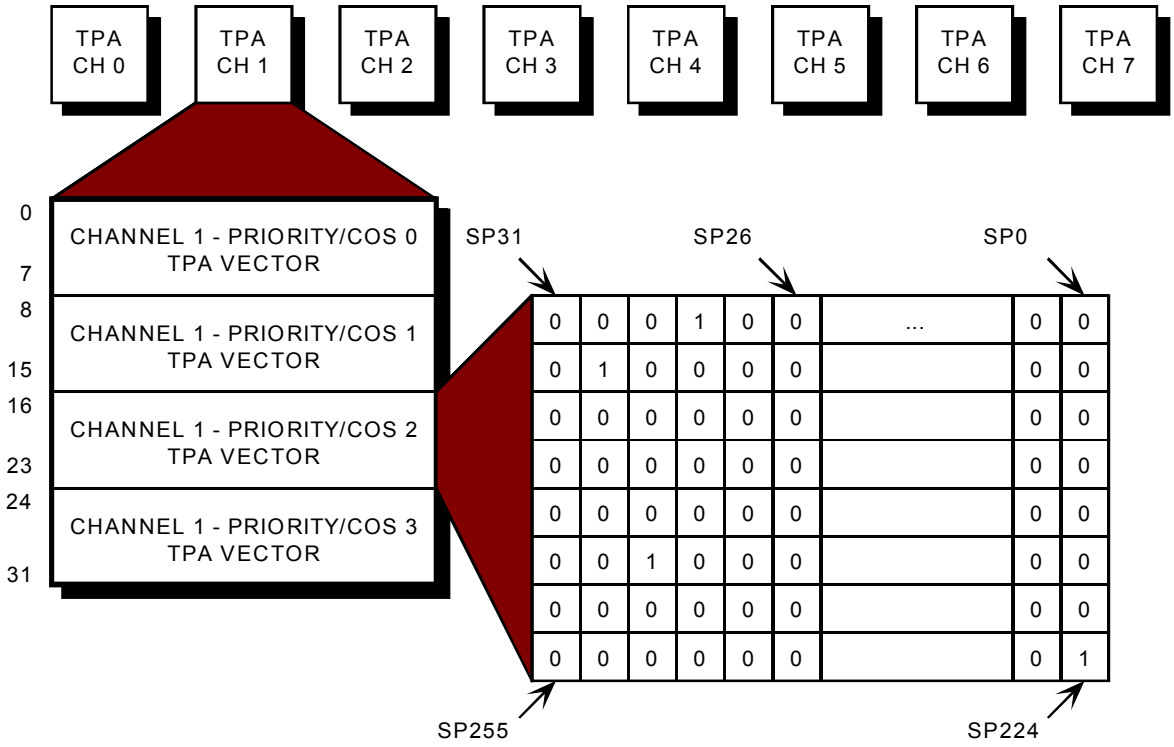

Figure 8. The Transmit Pending Arrays configured for 8 x 100 Mbps operation. Each bit indicates if a QA has been received from a given source/priority (or class of service) queue. When the QA request has been serviced, the bit is cleared.

A TPA bit is set following the receipt of a QA message for the given channel/source port/priority. A TPA bit is cleared when a DB message with an EOF indication is received for that channel/source port/priority.

The array shown above illustrates an example where destination channel 1, priority level 2, has received a QA for each of source ports 28, 62, 189 and 224.

3.3.2 1 Gbps mode

In 1 Gbps mode, there is a single Transmit Pending Array (TPA) for the single supported channels. The TPA consists of four vectors, one vector for each class of service. There is a 4-bit value in the vector for each source port. The 1 Gbps TPA arrangement is illustrated in Figure 9.

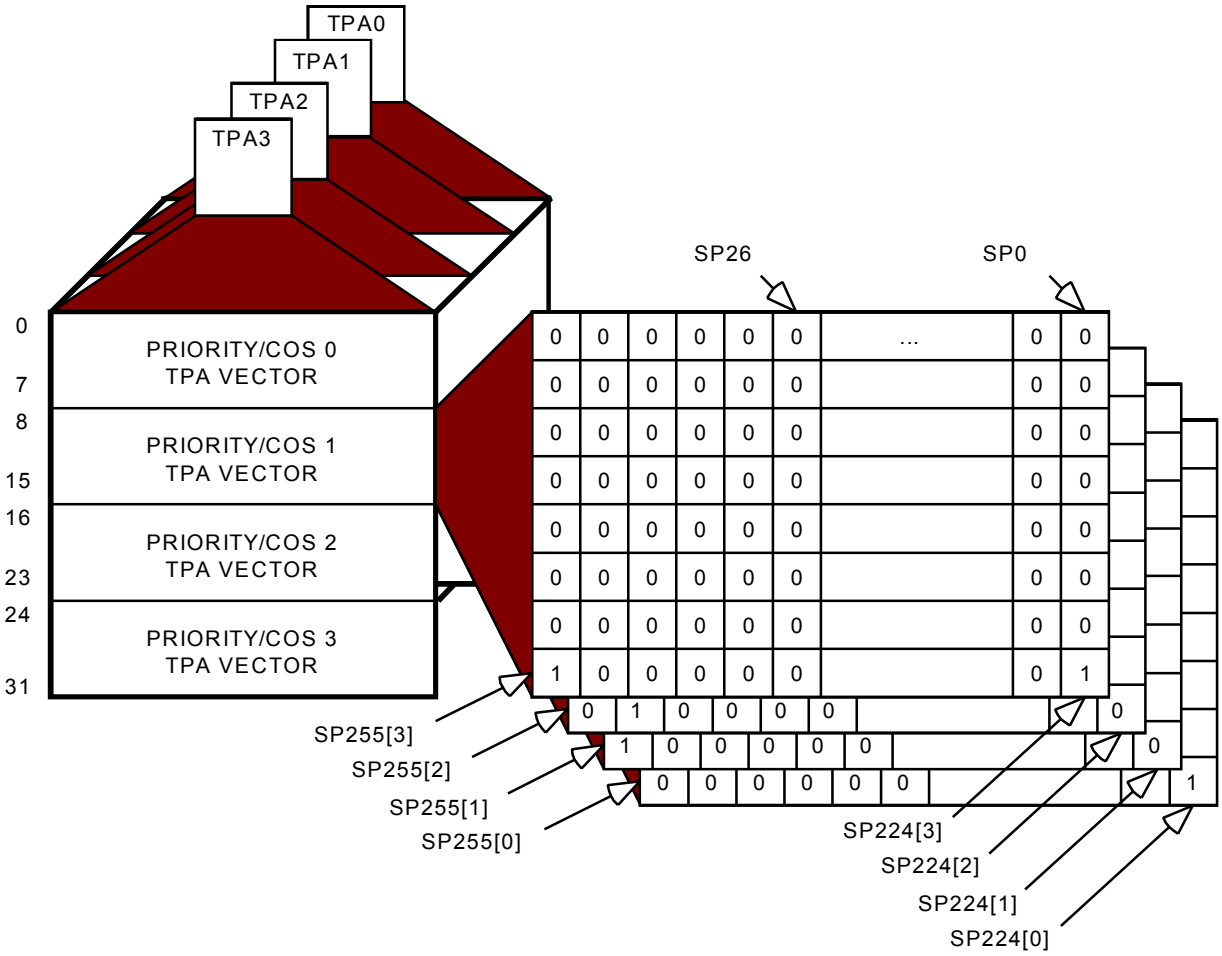

Figure 9. The Transmit Pending Arrays configured for 1 x 1 Gbps operation. Each 4-bit value indicates the number of unserviced QA messages which have been received from a given source/priority (or class of service) queue. When a QA request has been serviced, the corresponding value in the TPA is decremented. This figure illustrates an example where priority level 1 has received 9 QAs for SPID 224, 8 QAs for SPID 254 and 10 QAs for SPID 255.

A TPA count is incremented following the receipt of a QA message for the given source port/priority. A TPA count is decremented when a DB message with an EOF indication is received for that source port/priority.

3.3.3 Debug Access

The contents of the TPA RAMs can be accessed through the CRI for diagnostic purposes. The TPA number (0 through 7) and TPA address are programmed into the TPA Diagnostic Control Register (indirect registers). After 4 SYSCLK cycles after the TPA number and address have been written, the data word has been latched and may be read out of the TPA Diagnostic Data registers (indirect registers 0x21 and 0x22). This operation is transparent to the EQM's operation and does not require the EQM to be halted.

3.4 Queue Fetch Scheduler (QFS)

The Queue Fetch Scheduler is responsible for transmitting the QF messages as a result of detecting a pending request for data as indicated in the transmit pending array (TPA) for a given channel.

The format of queue fetch message as presented to the EXI is illustrated in Figure 10. Note that the EQM only supports 256 SPIDs and 8 DCIDs. The unusued most significant bits in each field (SPID[9:8] and DCID[5:3]) will always be zeros.

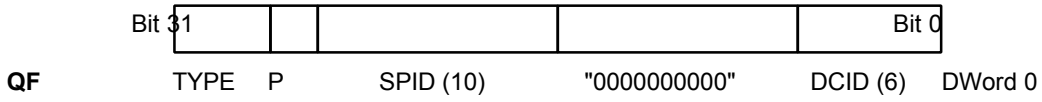

Figure 10. Queue Fetch (QF) Message Format. QF messages are sent by the EQM to indicate that it is ready to accept a single DB message from the queue identified by the SPID and P fields.

The description of the various fields in the QF message are given in Table 2:

Table 2. QF Message Field Definition.

Field	Bits	Description
TYPE	4	Message Type. Defines up to 16 possible message types, of which the EQM requires to know only 3, namely QA, QF, DB. 0000 - Data Block (DB) 0010 - Queue Allocate (QA) 0100 - Queue Fetch (QF) Other codepoints are reserved.
P	2	Priority of the frame. Every frame is allocated one of four priority levels. The IDMA passes this field on to the queue manager. The queue manager will queue the frame according to the following: '00' - priority 0 '01' - priority 1 '10' - priority 2 '11' - priority 3 Note that priority 0 is the highest priority and priority 3 is the lowest. The external CPU provides this field as part of the forwarding information.
SPID	10	Source Port Identifier This field indicates the EXACT ring source port from which the queue allocate was received. This field is returned as the destination port in a QF message.
DCID	6	Destination Channel Identifier This field indicates the destination channel for which the queue allocate or data is intended. The upstream EXI substitutes the absolute port ID (queue ID) with the channel ID using a base offset.

3.4.1 100 Mbps mode

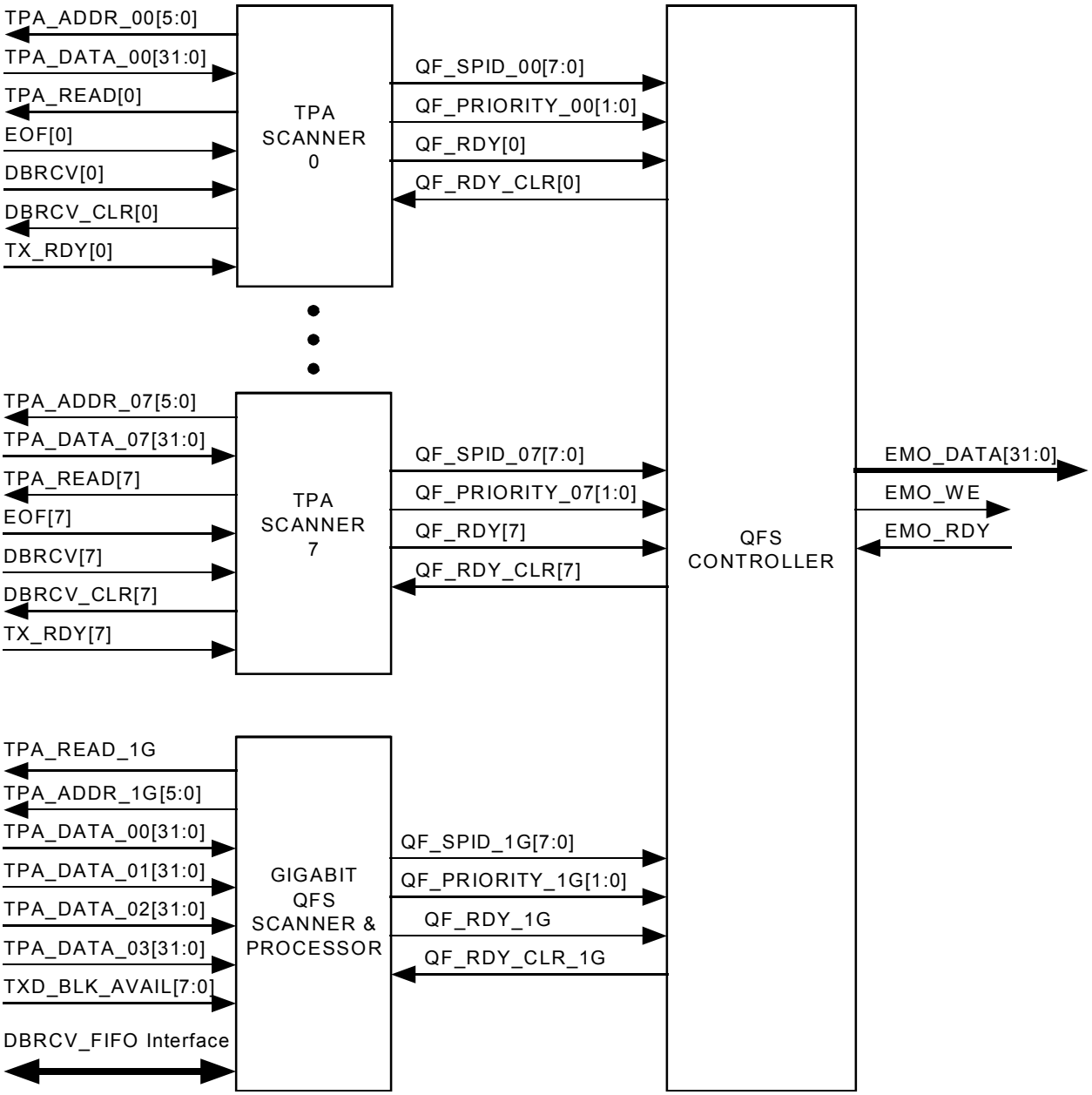

Figure 11. Queue Fetch Scheduler Block Diagram. The QFS consists of eight TPA scanners which search the TPAs for set bits in 100 Mbps mode, a single scanner and processor which searches the TPA for non-zero QA counts in 1 Gbps mode and a controller which arbitrates between the scanners and issues the QF messages.

The 100 Mbps QFS contains a TPA Scanner block for each channel supported. These scanners independently scan the associated TPA for set bits according the scheduling algorithm currently employed. (The TPA for each channel must be scanned independently and in parallel in order to meet the QA received to QF issued latency requirements). When a TPA Scanner detects a request for a data frame, it places the corresponding source ID and priority/class of service indication on its output and raises a flag. The QFS implements a round robin polling mechanism on the TPA Scanners and constructs the QF message when a TPA Scanner ready flag is detected, adding the relevant TYPE and DCID fields.

Once the QFS scheduling algorithm (see following section) has decided to process a given QA, the frame is processed as follows:

- Generate queue fetch for this channel, and send the QF to the indicated source port
- Wait until the QADBP sets the DBRCV (DB Message Received) flag for this channel.
- Clear the DBRCV and send another QF message for this channel, sending to relevant source port.
- Repeat until the QADBP sets the EOF (End of Frame) flag for this channel
- Return to scheduling algorithm

3.4.2 1 Gbps mode

The 1 Gbps QFS supports a single destination channel, but in order to reduce the QA -> QF -> DB latency, it supports multiple outstanding QFs on multiple SPID/P traffic streams. The QFS consists of a TPA Scanner, a DBRCV FIFO and a Queue Fetch Request Processor (QFRP) as shown in figure 11.

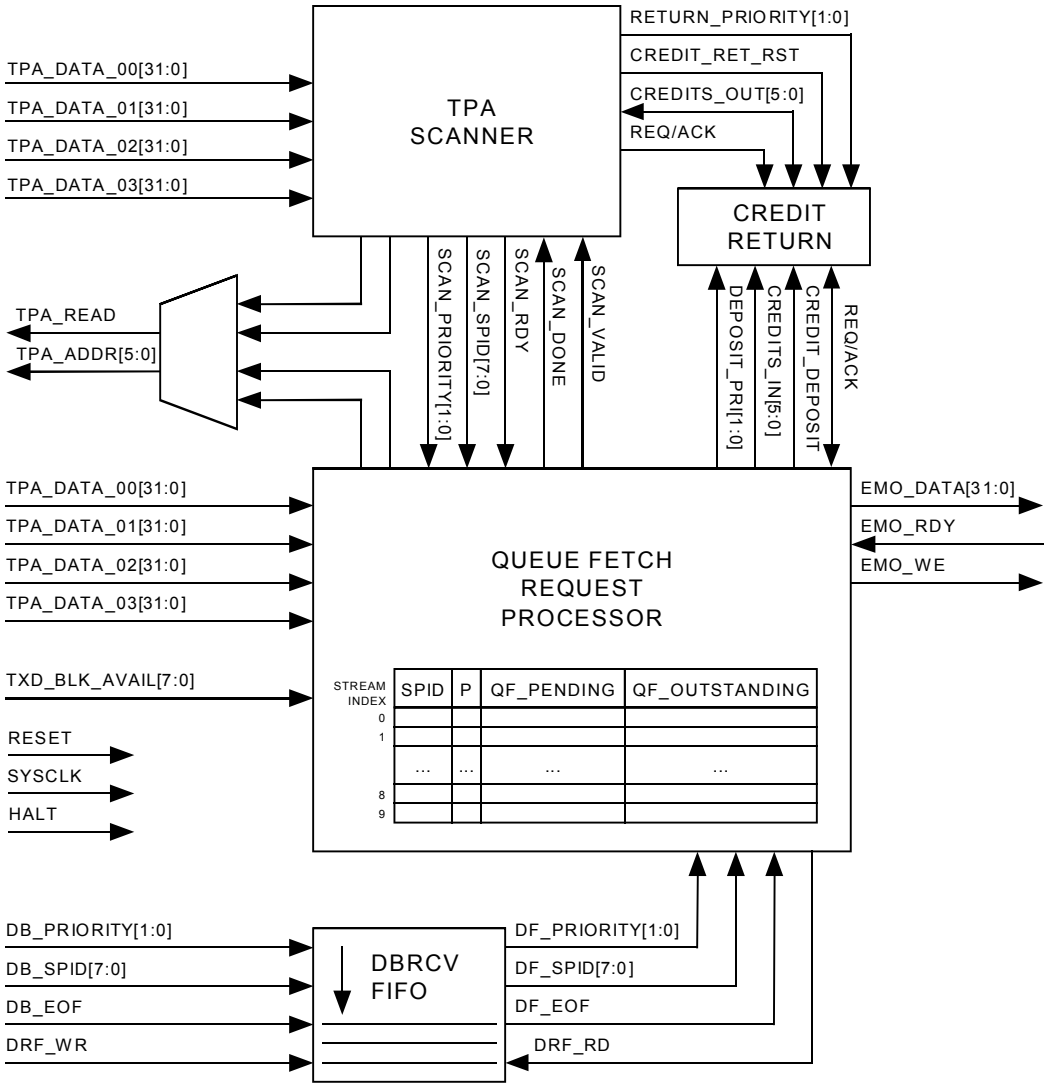

Figure 12. Queue Fetch Scheduler: 1 Gbps Scanner and Processor Block Diagram. The QFS scans the TPA for non-zero entries and issues QF messages according to the scheduling algorithm selected and tracks the QFs pending and outstanding for each data stream.

The QFRP keeps track of the state for each of the 10 streams using a state vector consisting of the SPID/P assigned to that stream, the current number of QFs pending (i.e., assigned but pending a check of space available in the TXCTRL Frame Reassembly Buffer) and QFs outstanding (i.e., QFs which have

been issued after checking that there is sufficient space available for the resultant DB payloads).

The single TPA scanner operates very similar to the 100 Mbps mode and uses the same scheduling algorithms; however, it no longer stops and waits for an entire frame to be processed when a non-zero TPA location is encountered in the course of searching the array. Instead, the TPA Scanner passes the SPID/Priority pair to the Queue Fetch Request Processor along with the value it read form the TPA.

The QFRP will check if the SPID/P combination matches one of the data streams currently being serviced. If there is no match, the QFRP will assign the SPID/P to an unused stream if available. If no unused stream is available, then the QFRP will continue processing but will not complete the handshake with the TPA Scanner, effectively stopping scanning until a free stream is available. If the SPID/P matches the SPID/P assigned to an active steam, then the current number of outstanding (issued) QFs for the stream and the number of QFs for that same stream which are pending are checked. If the TPA count is greater than the SUM of the pending and outstanding QFs, the QFRP will increment the QF pending count and indicate that the request from the scanner is valid by asserting VALID and SCAN_DONE; otherwise VALID is not asserted when the SCAN_DONE indication is asserted.

If the TPA Scanners sees that its' request was valid, it will continue the scanning algorithm as in the 100 Mbps mode, treating the VALID indication as if the frame had be completely processed. If the request was not valid, then the scanner continues on as if the TPA count for the current SPID/P had been zero.

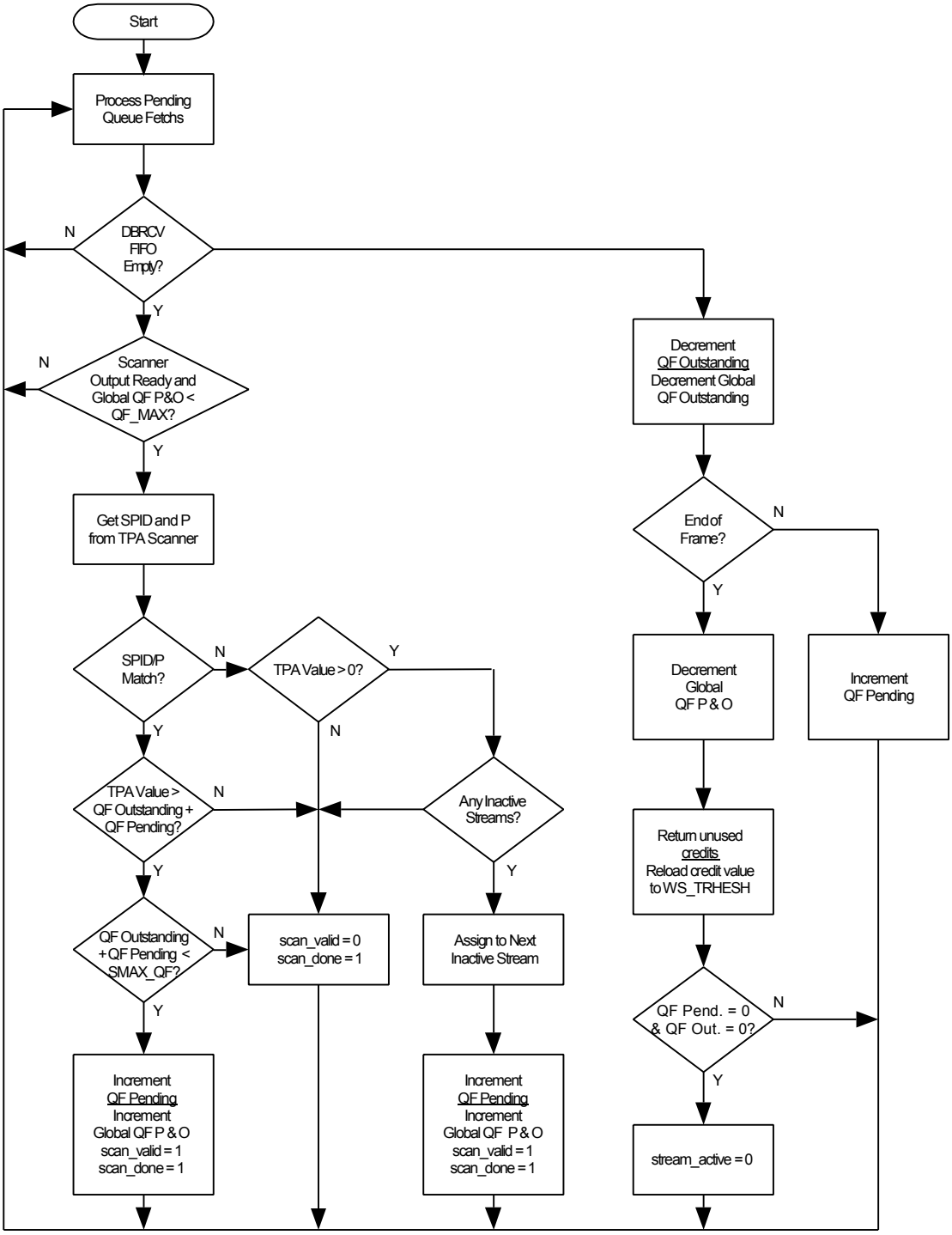

Figure 13. Queue Fetch Scheduler Algorithm: 1 Gbps. The Queue Fetch Scheduler has a independent scanner operating in parallel which uses the same algorithms as the100 Mbps schedulers; except that it immediately starts scanning after the decision is made to process a frame.

The Queue Fetch Request Processor also services requests from the DBRCV FIFO. For each entry processed from this FIFO, the stream QF Outstanding count is decremented. If the EOF flag is not set for the entry, the stream QF Pending count is incremented. If the EOF flag is set, the QFRP checks if both the stream QF Pending and QF Outstanding counts have been reduced to zero. In this case, the stream is marked inactive and subsequently freed up for use by another SPID/P pair.

The QFRP also processes the QF Pending counts. For each stream, the QFRP checks if QF Pending is non-zero and if so checks if there is uncommitted space available in the associated frame reassembly buffer. If space is available, the stream's QF Pending count is decremented, the QF outstanding count is incremented and the QF is sent to the EXI.

The QFRP also restricts the maximum number of outstanding QFs across all streams to the value programmed in the Global Maximum QF Outstanding Register. For the PM3380 port controller, a nominal value of 10 is recommended based on system level simulations done by PMC-Sierra's chip architect, Tom Alexander. However, the value may be changed to optimize performance in other system with different fabric latencies. The value may be set as high as the maximum number of memory blocks in the TXCTRL Frame Reassembly Buffer (i.e., 70 for block size of 120 bytes, 140 for block size of 240 bytes). Setting GMAX_QF larger than the number of blocks in the TXCTRL FRB will not permit more QFs to be issued.

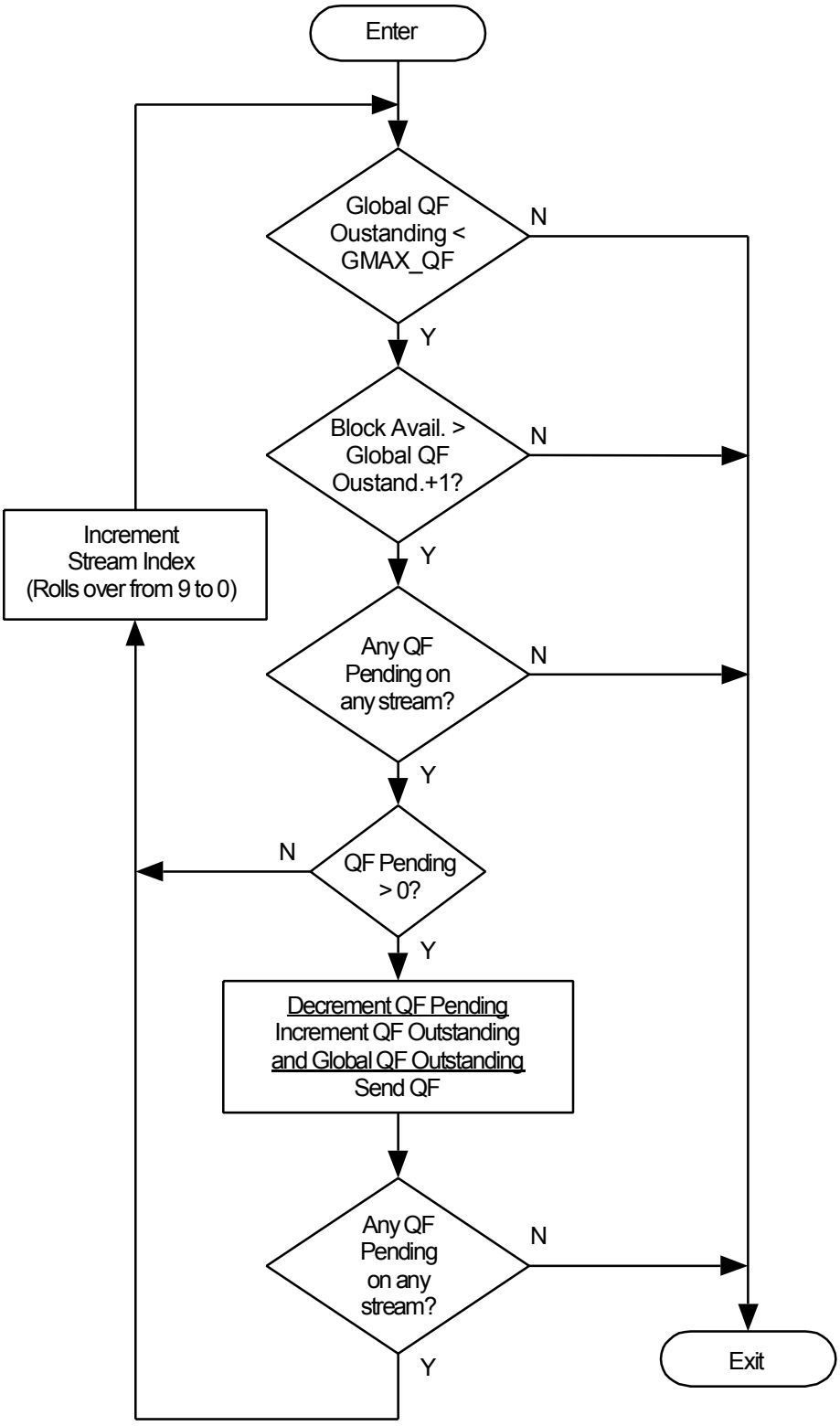

Figure 14. Queue Fetch Pending Processing: 1 Gbps mode only.

3.4.3 Scheduling Modes

The QFS implements three modes of scheduling:

1. Strict Priority Scheduling (SPS)
2. Weighted Priority Scheduling (WPS)
3. Weighted Fair Scheduling (WFS)

The scheduling mode is selectable by the SCHMODE[1:0] field in the EQM General Control Register.

3.4.3.1 Strict Priority Scheduling

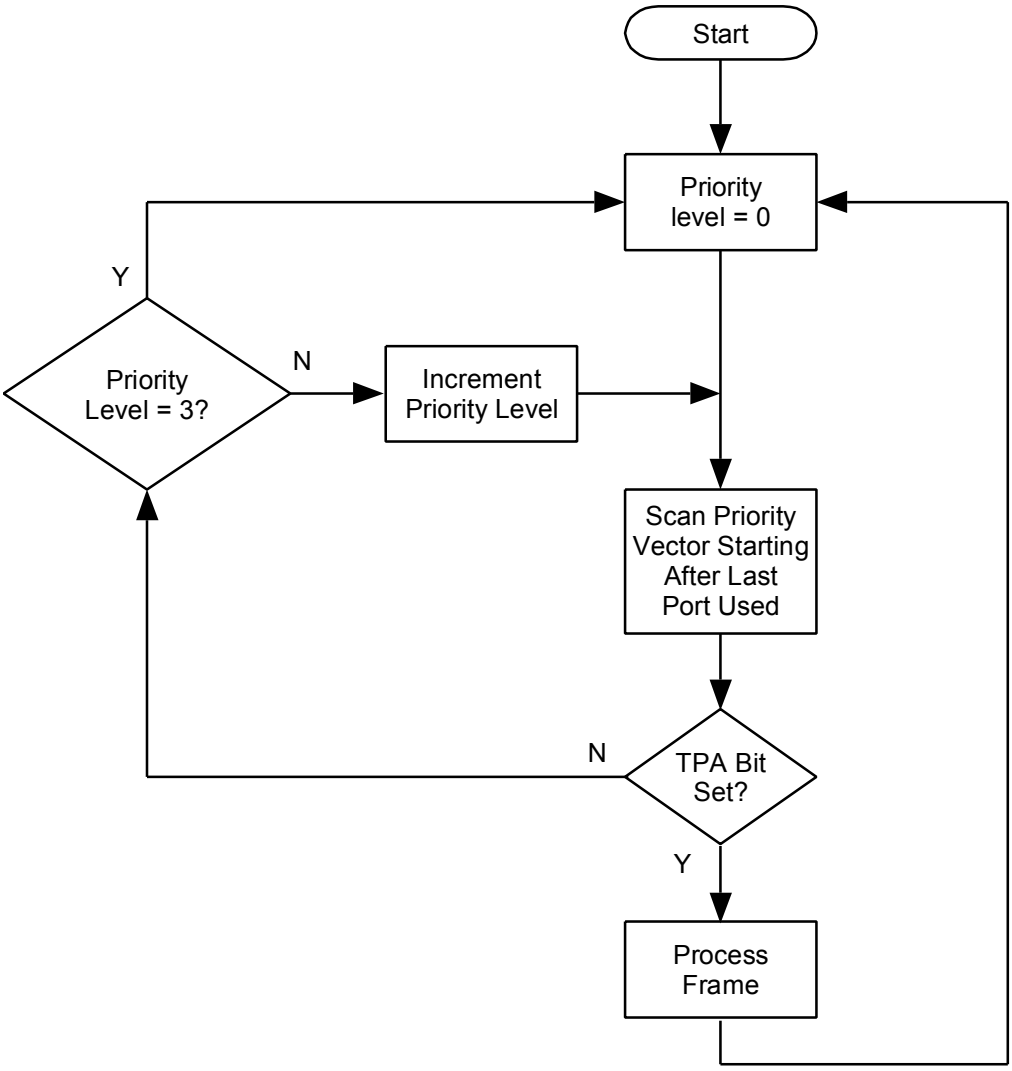

Figure 15. Strict Priority Scheduling Algorithm. After processing each frame, SPS always checks to determine the highest priority level which has outstanding QA messages. Round robin order is maintained within each priority level. If enough traffic is presented to the high priority queues, lower priority queues can be starved.

Strict Priority is a simple scheduling method. For any given egress channel, the scheduler begins by scanning the transmit pending array (TPA) for the top priority level beginning after the index of the last source port which was serviced from this priority level. If the TPA indicates that frame transmissions requests are pending, then the first such request is serviced in source port order, until the

entire data frame has been received. Additional frame requests at the top priority level are processed until all requests have been exhausted, then the vector for the next lowest priority is scanned. If there are no transmit requests pending for this priority level, the scheduler will proceed to the next lowest priority level, and repeat the scanning process. If there are transmit requests pending for this priority level, the next eligible transmit request (i.e., the request for the next port after the last port serviced) is processed and then the priority level is reset to the top priority.

3.4.3.2 Weighted Priority Scheduling

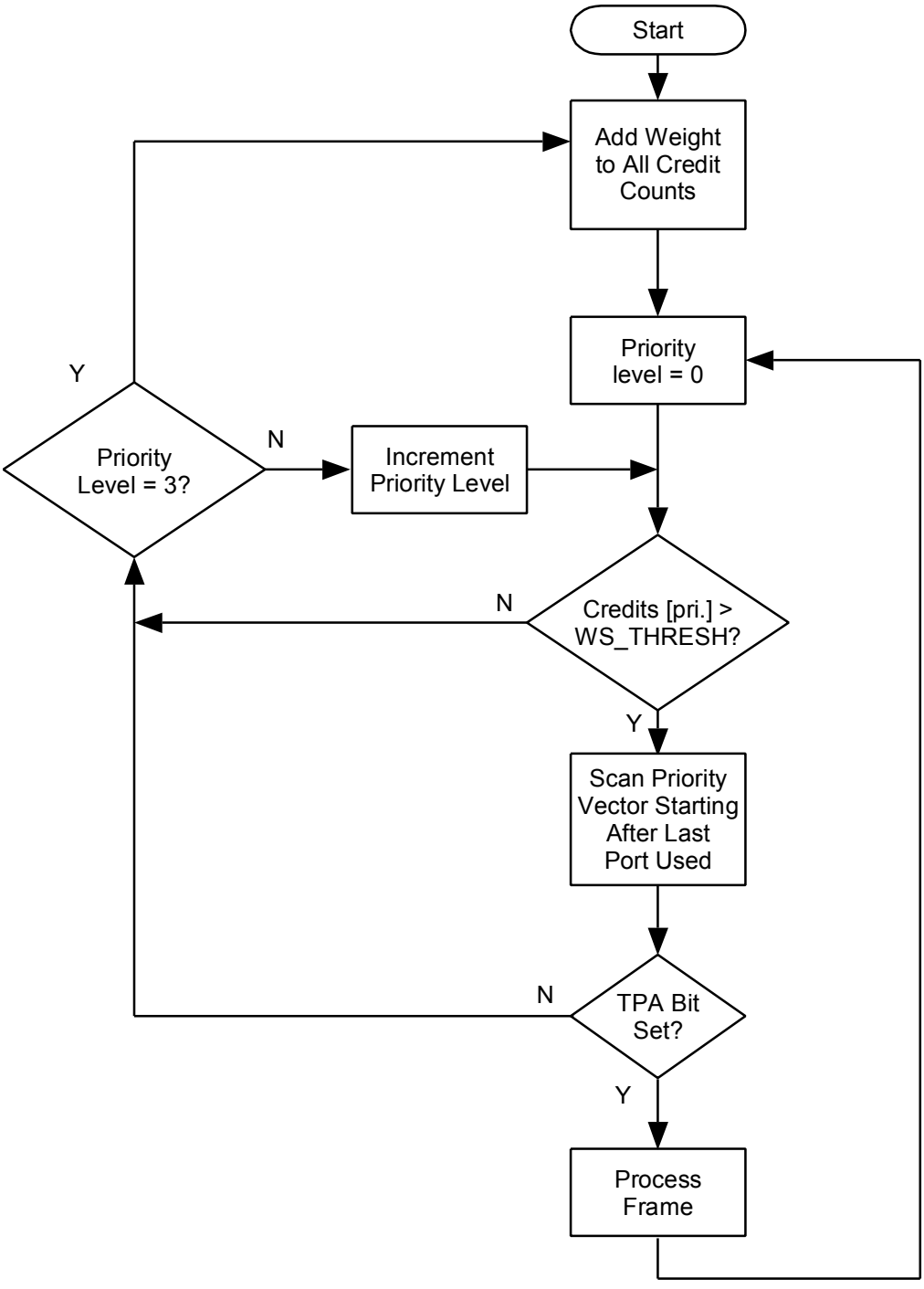

Figure 16. Weighted Priority Scheduling Algorithm. WPS acts as a priority scheduler with credit limits to prevent starvation of lower priority queues.

The Weighted Priority Scheduling mechanism provides the ability to distribute source data traffic over 4 weighted classes of service and prevent starvation of lower priority levels, while still maintaining a priority hierarchy.

Weighted Priority Scheduling (WPS) is implemented using a credit accumulation scheme and weighted priority servicing, requiring a total of three storage registers (weighting, credit store and credit limit) per priority level and one adder/subtracter per channel. For any given egress channel, the scheduler begins by adding the weighting value to the respective data block credit store for all priority levels. If sufficient credits are available for the highest priority level to transmit a maximum sized frame (i.e., the credit store is >= WS_THRESH) then the transmit pending array (TPA) is scanned for that priority level beginning after the index of the last source port which was serviced from this priority level. If the TPA indicates that frame transmission requests are pending, then the first such request is serviced in source port order, with the credit store being decremented for each data block requested until the entire data frame has been received. Additional frame requests at the top priority level are processed so long as the credit store remains above the threshold level. When all top priority transmit requests have been exhausted or the credit store has been reduced below the threshold, the credit store for the next lowest priority level is checked and if its credit store is above the threshold level, the TPA is scanned. If there are insufficient credits available or there are no transmit requests pending for this priority level, the scheduler will proceed to the next lowest priority level, otherwise the next eligible transmit request (i.e., the request for the next port after the last port serviced) is processed and the priority level is reset to the top priority.

When all priority levels have either a credit store value below the threshold level or no transmission requests pending, the scheduler again updates the

credit stores for all priority levels by adding the weighting value to the value remaining in the data block credit store. The credit store registers saturate at the limit value. The scheduler then begins again at the highest priority level.

Note that if the limit value is set to the weight value, the scheduler will be memoryless and will allocate the same share of scheduling requests each time through the loop as specified by the weighting values. However, if the limit value is larger than the weight value, then that given priority level may store its unused bandwidth up to the credit limit and the weight values become time averaged weightings. (A limit value lower than the weight value is an illegal condition since it would effectively become the weighting value).

3.4.3.3 Weighted Fair Scheduling

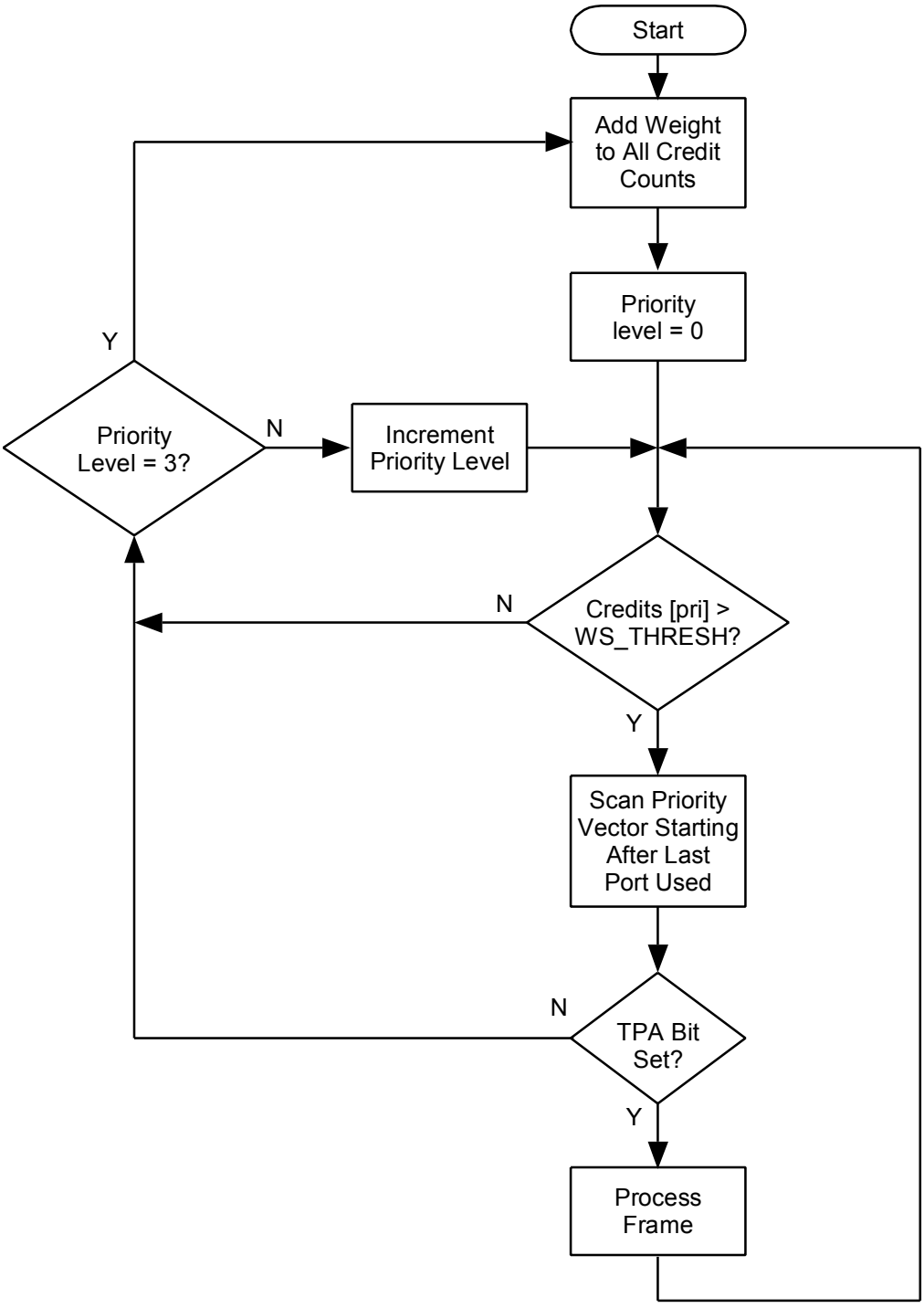

Figure 17. Weighted Fair Scheduling Algorithm. The WFS algorithm is very similar to the WPS algorithm: the only change is that after processing a frame, the flow jumps to the credit check instead of resetting the priority to zero as in WPS.

Weighted Fair Scheduling mechanism provides the ability to distribute source data traffic over n weighted classes of service.

Weighted Fair Scheduling (WFS) is implemented using a credit accumulation scheme and weighted round-robin servicing, requiring a total of three storage registers (weighting, credit store and credit limit) per class of service and one adder/subtracter per channel. For any given egress channel, the scheduler begins by adding the weighting value to the respective data block credit store for all classes of service. If sufficient credits are available for the first class of service to transmit a maximum sized frame (i.e., the credit store is >= WS_THRESH) then the transmit pending array is scanned for that class of service beginning after the index of the last source port which was serviced from this class of service. If the TPA indicates that frame transmission requests are pending, then these requests are serviced in source port order, with the credit store being decremented for each data block requested until the entire data frame has been received. Additional frame requests in this class of service are processed so long as the credit store remains above the threshold level. When all transmit requests have been exhausted or the credit store has been reduced below the threshold, the next class of service is processed. When all classes of service have been serviced, the scheduler again updates the credit stores for all classes of service by adding the weighting value to the value remaining in the data block credit store. The credit store registers saturate at the limit value.

Note that if the limit value is set to the weight value, the scheduler will be memoryless and will allocate the same share of scheduling requests each time through the loop as specified by the weighting values. However, if the limit value is larger than the weight value, then bursting is allowed for that given class of service and the weight values become time averaged weightings. (A limit value

lower than the weight value is an illegal condition since it would effectively become the weighting value).

The WFS mechanism is a completely fair system which does not assign any priority to a particular class of service and simply acts as a means of sharing the available bandwidth for a channel between different class of service without wasting unused bandwidth the way traffic shapers or other bandwidth reservation systems may.

3.5 Transmit Frame Reassembly Buffer Interface (TFRBI)

The Transmit Frame Reassembly Buffer Interface is responsible for transferring the payload data from the DB message to the relevant TXCTRL Frame Reassembly Buffer. Only the payload data is transmitted to the downstream FIFO. (Note: the TXSTAT field from the DB header is placed on the TXCTRL interface TXD_STAT pins at the time the payload data is transferred). The DB message header field is stripped from the data stream by the QADP. The Frame Reassembly Buffer is configured as eight virtual FIFOs for the 100 Mbps mode and as a single large pool for reassembling frames from up to ten traffic streams in the 1 Gbps mode.

The TFRBI maintains per channel/stream counts of the bytes as they are assembled in the MAC FIFOs. At the end of the complete frame, the TFRBI writes this information to the Tx Statistics Transaction FIFO.

The TFRBI does not check the status of the TXD_BLK_AVAIL/TXD_RDY bus since these signals are checked by the QFS before issuing a QF.

3.5.1 Transmit Statistics Collection

The EQM maintains the following transmit statistics for the each destination MAC address.

- Transmit frames total count
- Transmit bytes total count

Statistics counts are maintained in external SSRAM. The EQM receives a 16-bit pointer in each of the Data Block (DB) messages (TXSTAT) to indicate the index of the particular statistics data structure.

The update of the transmit statistics counters depend on the state of the transfer of data to the downstream Tx MAC FIFOs, after the last data block has been transmitted. Note that if TXSTAT=0, then no counters are updated. Note that the counters do not saturate but will rollover, and that it is the responsibility of system software to maintain counter integrity.

Each data structure consists of two 32-bit double words (Dwords) and contains two 32-bit counts. The TXSTAT index is used with the TXST_BASE offset control register and the particular statistics counter being accessed, and together form a 22-bit address into external SSRAM. This is illustrated in the SUMI description later.

The various counters and their update conditions are illustrated in below:

Table 3. Transmit Statistics.

Field	Bits	Description	Update Conditions
TX_FRM	32	Transmitted Frame Total	Updated at the end of the last data block if no error conditions detected.
TX_BYTES	32	Transmitted Bytes Total (Running total of bytes transmitted to the downstream TxCTRL.)	Updated at the end of the last data block if no error conditions detected.

When the last DB for a given frame is transferred, the TFRBI places Transmit Statistics Update Transaction (TSUT) into the Tx Statistics Transaction FIFO (TSTF). The format of the TSUT message is as follows:

Figure 18. Transmit Statistics Update Transaction (TSUT) Format. The TSUT is passed from the TFRBI to the SUMI after the last byte of a frame has been passed through.

3.6 Transmit Statistics Transaction FIFO (TSTF)

The 32 entry Transmit Statistics Transaction FIFO acts as a buffer between the TFRBI and the SUMI. The TFRBI may need to capture multiple TSUTs in a very short period of time. For example, consider the extreme case where a single byte DB with EOF set for each channel is received back to back. While the SUMI module can process requests fast enough to keep up with full line rate traffic, it needs to have bursts of TSUTs buffered so that no requests are lost due to the SUNI's latency in accessing the MPAC.

3.7 Statistics Updater and MPAC Interface (SUMI)

The Statistics Updater and Memory Port Access Controller Interface processes the TSUT messages from the TSTF and permits the EQM to access the transmit statistics structures which are held in an external memory:

The Transmit Statistics Data Structure is stored in external memory as shown in Figure 21. There may be up to a total of 65,535 data structures. Each structure contains two 32-bit Dwords, as illustrated in Figure 19. The two Dwords are running counts of the total frames and total data bytes respectively passed through the EQM to the downstream transmit frame reassembly buffer.

Figure 19. Transmit Statistics Data Structure. The 2-Dword structure tracks the total number of frames and bytes transmitted in packets containing the corresponding TXSTAT field. The statistics reflect a running count, which must be read frequently enough by an external CPU to prevent the counters from wrapping without being read.

The data structures accessed by the SUMI and their respective addressing is illustrated in Figure 20 and Figure 21 respectively. This illustrates the case where a single 16M-bit SSRAM is used externally.

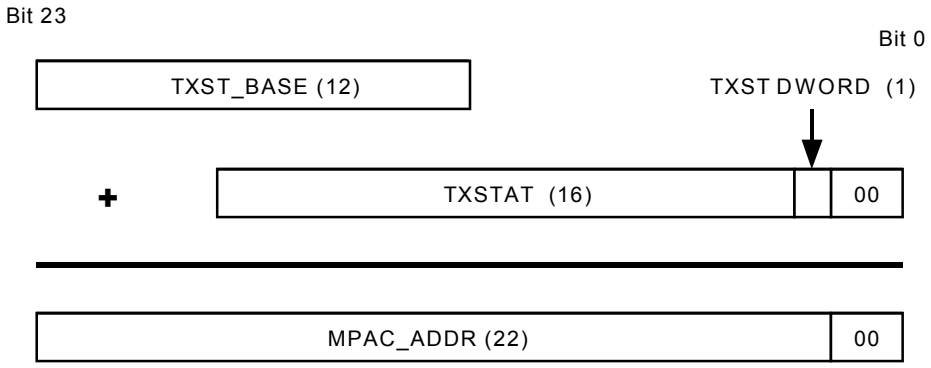

Figure 20. SRAM Data Structure Addressing. The TXSTAT value from the TSUT is used to index into the memory block specified by TXST_BASE.

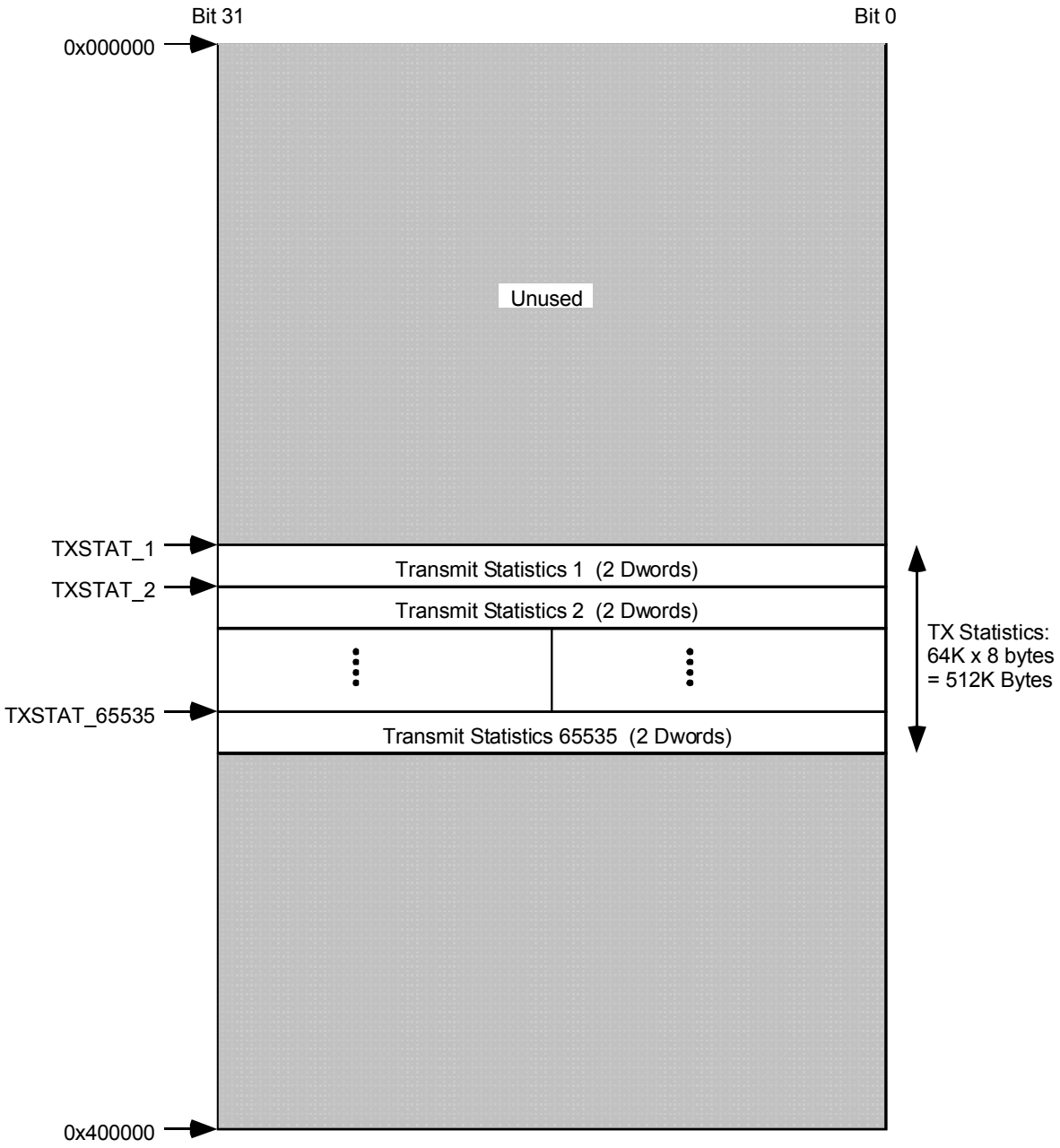

Figure 21. Example SSRAM Memory Map with 2 MByte used by EQM. The unused portions of the memory are used by other blocks in the port controller for queue structures and statistics.

4 Simulations

4.1 Functional Simulations

The functional simulations for the EQM were performed by selecting the appropriate Verilog modules in one of two test benches. These entities control/monitor the EQM I/O, and generate output files which can be checked for correct operation

The first test bench operates in a simple open loop mode, which presents individual QAs and DBs to the EQM after specified delays. This mode is used for simple tests, tests which require a specific sequence of operations, and tests for error conditions.

The second test bench implements source queue generators for each of the potentially 1024 SPID/P combinations. This mode reads in a list of frames for each source with DCID, frame length and earliest transmission time. The source mimics the behavior of an IQM block, sending QAs when the transmission time is reached and responding with DBs to each QF. When all of the scheduled frames for a given source have been transmitted, that source signals that it is finished, when all sources have finished, the test terminates. While this mode does not provide output checking, it does provide an automatic check of the EXACT protocol.

The results of simulations made with the second testbench can be analyzed using a perl script, which calculates the average BW percentage allocated to each priority (or class of service level) over the period in the simulation during which all streams at all priority levels are active.

4.2 Performance Simulations

A suite of simulations was created to test the performance of the EQM's WFS and WPS scheduling modes. All of these tests are run using testbench 2. The tests were carried out using varying numbers of streams (4 to 32), weights (equal ratios, arithmetic ratios and geometric ratios), limit-to-weight ratios (2:1, 3:1, 4:1), and frame sizes (64 byte tinygrams, 1522 byte maxigrams, n*240 byte, and random) in both 100 Mbps and 1 Gbps modes. A complete list of the specific simulations executed and results are in Appendix B.

In general the EQM WFS and WPS scheduling modes were able to distribute bandwidth between the priority (class of service) levels within an error bound of +/- 2.0% of the total bandwidth. There were a few data points which lie outside this error bound; however, it was judged that the traffic distribution was accurate enough for the intended applications.

5 Synthesis

The EQM design was implemented in Verilog HDL code and then synthesized into a gate level netlist using Synopsys Design Compiler. The total gate count of the EQM exceeded 70,000 gates, which was a large block design for the 0.35 micron technology used to implement the port controller ICs. Theoretically, Design Compiler should have been able to synthesize the entire EQM design in a single operation. In practice the design proved to be challenging for the synthesis tool: runs took multiple days to complete and the tool was unable to meet the timing requirements. Consequently, each sub-module (including top-level glue) was individually synthesized, then the sub-modules were assembled in a separate operation (no synthesis) and the overall timing confirmed.

The hierarchical synthesis approach yielded several benefits. First, each block was small enough that Design Compiler would complete a synthesis run in a matter of hours (instead of days). Secondly, the 100 Mbps QF scheduler could be synthesized once and then instantiated 8 times (rather than being synthesized 8 times in a top level synthesis approach). Finally if the synthesis program had difficulty meeting timing objectives for any given sub-module, it was easier to identify where the problem occurred, and make adjustments either to the sub-module architecture or the individual synthesis script and resynthesizing only the sub-module and its dependent hierarchy rather than resynthesizing the entire design.

Hierarchical synthesis did have one problem. Due to the cycle timing requirements of the design, it was not possible to place registers on the boundaries of each sub-module. Therefore, time budgeting had to be accomplished on the outputs and inputs of some sub-module. In particular, this

tended to occur when there was a large multiplexer on the output of a sub-module feeding into combinatorial logic on the input of another sub-module. Synthesis could have been made much easier by putting the multiplexer logic into the target sub-module, leaving the source sub-module with registered outputs. However, this approach would have created physical routing problems with hundreds of metal layer traces needing to be routed between blocks. Consequently, some significant effort was required to balance the timing budget between the sub-blocks. Synthesizing the EQM as a flat entity would have removed this issue, but as stated earlier, this approach had even more significant problems.

6 Design reusability

The EQM was designed with reusability in other IC designs as a priority. Wherever possible, the Verilog code was parameterized so that that it would be easy to expand the number of ports. The design was broken into sub-blocks that could easily be reused to increase the number of ports serviced. Simulation and synthesis scripts followed the same approach. The TPA memories were designed as separate sub-modules so that a change could be easily made from SRAM to register files if the design was re-implemented in a smaller geometry silicon technology.

The EQM was designed such that all inputs and outputs to the block are registered. Furthermore, the internal configuration and control registers are all accessed by a standard PMC-Sierra on-chip bus. These features (used on all PMC-Sierra IC designs) make reuse of the silicon building blocks in new IC designs much easier.

7 System performance

The scheduling system did operate as intended on the system level. However, architectural decisions made in the design of the ingress queue manager (IQM) block reduced the usefulness of the scheduling algorithms. The IQM in 8-channel (100 Mbps) mode maintains a single queue for each destination EQM in the system. Therefore, traffic from all 8 input ports is placed on a single queue for each destination and priority (or class of service) in a first come first served basis. Consequently, while the EQM served ports using round-robin fair queuing within a class of service, the merging of traffic from multiple ports within the IQM reduced the effectiveness of this approach. Creating queuing structures for each input port would have been prohibitive in terms of the logic required in the IQM given the restrictions on design size.

The merged queuing issue did not affect SPS and the ability to implement IEEE 802.1p. In this case, the incoming packets of the same priority level were simply queued at the IQM in a first-in-first-out approach, which is an acceptable means of scheduling this traffic.

8 A Comparison of Fair Queuing Techniques

The EQM's technique of arbitrating traffic is the most technologically advanced part of the design. The ability to provide higher priority to different classes of packets is useful to allow time sensitive data (e.g. data required to dynamically monitor and configure the network, or real-time voice and video transmission) to jump ahead of less time sensitive data (e.g. e-mail and file transfers). The IEEE 803.1p standard establishes a simple two level priority scheme. Similarly, it may be useful to somehow allocate the available bandwidth of a port among several sources either equally or with a non-equal weighting. This feature can be used to prevent a given source from unfairly monopolizing the port bandwidth. Finally, the two techniques are combined in the WPS scheduling method, which gives a given class of service priority until it exceeds its maximum allocated bandwidth, which prevents the starvation of lower priority queues.

In the consideration of the EQM's scheduling algorithms, it is worthwhile to note that the goal was not to produce a traffic policer, which would restrict the bandwidth of a given source (either class of service or individual source ports). The goal was to use all of the available bandwidth, but to provide flexibility in allocating access to the available bandwidth by competing sources.

Previous techniques of implementing fair queuing algorithms described in the literature, while not specific in implementation, were generally more suitable for processor/software based designs. While these designs could have been implemented directly in an integrated circuit, the cost in terms of silicon area would be prohibitive. It should also be noted that focus on queuing algorithms in the literature is generally based on allocating bandwidth between individual packet sources. The focus for the EQM is allocating bandwidth on the basis of

four priority levels or classes of service, regardless of the source of the packets. Indeed, within a given class of service, the EQM serves all packets on a round-robin basis, which is the most basic form of fair queuing. (As mentioned in section 7, the performance of the queuing system is compromised by the effecting merging of 8 input ports into a single traffic source for a given class of service by the IQM architecture). The more advanced queuing algorithms are applied at the class of service level, allowing for paramaterizable bandwidth distribution between service classes.

Nagle first proposed a fair queuing algorithm for datagram networks in 1987 [6]. Prior implementations used a single queue into which all packets were placed on a first come first served basis. Nagle's approach was to maintain separate queues for each packet source and then service these queues in a simple round robin fashion. If each source presented enough traffic to prevent its queue from emptying, then each source will have an equal number of packets serviced by the queuing device (a switch, router, or other gateway device). Nagle's approach failed to make any provision for packet length; therefore, while quite suitable for fixed packet networks, such as ATM, it would not provide "fair" results for variable length packet networks such as Ethernet or IP switches.

The EQM uses Nagle's fair queuing approach within a given class of service. Within a service class, source (IQM) queues are served on a round robin basis.

Demars, Keshav and Shenker's article from 1990 [4] extends Nagle's approach by taking into account the time to transmit each packet as well as the packet arrival time. This algorithm does a good job of fairly allocating bandwidth, as opposed to packet transmission, but requires significant computation to implement. Demars et. al. also suggest that their fair queuing algorithm can be

enhanced by allowing “arbitrary bandwidth priorities” to be assigned to given sources, which has become know as weighted fair queuing.

Credit Based Fair Queuing was introduced by Bensaou, Chan and Tsang [7] in 1997 as a practical implementation of weighted fair queuing. CBFQ sorts the queues using the following metric and services the queue lowest value:

$$\frac{packet\_size - credits}{bandwidth\_share}$$

This approach reduces the complexity of earlier implementations, which relied on virtual clocks; however, it still requires two computations including a division operation. The weighted fair scheduling technique implemented in the EQM disregards packet size when making the determination of which packet to send next. The bandwidth share is taken into account by varying the number of credits granted to a queue (in the EQM’s case, a class of service) each service round.

During each service round, new credits are added to the credit accumulation for each class of service subject to a programmable upper credit limit. Then packets within a given the first class of service are served, with a single credit being deducted for each EXACT data block message (up to 240 bytes) received. The class of service category is served with input ports selected in a round robin fashion until the credits are exhausted or the queue is empty. Service then proceeds to the next class of service queue. This is continued until all queues have been served, then a new service round begins and new credits are granted. This approach yields burstier results than does CBFQ, and the credit resolution is grainier, but the implementation is simple enough to be realized in silicon and the results (see appendix) were within acceptable error bounds.

9 Conclusions

The Egress Queue Manager was successfully used in two Ethernet port controller chip designs. The EQM allowed for IEEE 802.1p priority queuing operations as well as providing two weighted fair queuing modes achieved in a small hardware based design with low request (QA) to grant (QF) latency. While the overall system level queuing performance was compromised by architectural choices in the upstream ingress queue manager, the scheduling logic used in the EQM could be successfully used in other implementations. The scheduling design of the EQM was considered novel enough, that PMC-Sierra submitted a patent application for the design.

Appendix A: EQM Normal Register Descriptions

The EQM contains two register sets selectable by the Test Resister Select (TRSB) pin. When TRSB = 0, test registers may be accessed through the CRI bus. When TRSB = 1, normal mode registers may be accessed through the CRI bus.

The normal mode registers contain operation and configuration controls as well as device status indications. In most applications, the configurations registers will be programmed only at start-up or after a major mode change. Some applications may require dynamic provisioning of the parameters for the weighted scheduling modes, but otherwise no interaction with the register set should be required on a regular basis while operating.

After a reset, the register bits will assume the default values listed in each register description. An “X” in the default value listing indicates that there is no default value for the given bit. When writing to unused bits, the default value should always be written in order to avoid problems should a future version of the EQM assign a function to a currently unused bit location.

Table A. 1. Register 0x00: Indirect Register Select.

Bit	Type	Function	Default
Bit 15	R	IRDY	1
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R	Unused	0
Bit 4	R/W	REGSEL[4]	0
Bit 3	R/W	REGSEL[3]	0
Bit 2	R/W	REGSEL[2]	0
Bit 1	R/W	REGSEL[1]	0
Bit 0	R/W	REGSEL[0]	0

IRDY

The indirect ready register bit indicates that the Indirect Data register is ready to be read and that any write operations to indirect registers have been completed.

REGSEL[4:0]:

The register select bits allow the external CPU to access the indirect registers. Any data written to or read from Indirect Data register (address 0x1) will be to/from the indirect register addressed by the value of REGSEL[4:0].

Table A. 2. Register 0x01: Indirect Data.

Bit	Type	Function	Default
Bit 15	R/W	DATA[15]	0
Bit 14	R/W	DATA[14]	0
Bit 13	R/W	DATA[13]	0
Bit 12	R/W	DATA[12]	0
Bit 11	R/W	DATA[11]	0
Bit 10	R/W	DATA[10]	0
Bit 9	R/W	DATA[9]	0
Bit 8	R/W	DATA[8]	0
Bit 7	R/W	DATA[7]	0
Bit 6	R/W	DATA[6]	0
Bit 5	R/W	DATA[5]	0
Bit 4	R/W	DATA[4]	0
Bit 3	R/W	DATA[3]	0
Bit 2	R/W	DATA[2]	0
Bit 1	R/W	DATA[1]	0
Bit 0	R/W	DATA[0]	0

DATA[15:0]:

Any data written to or read from this register will be to/from the indirect register addressed by the value of REGSEL[4:0] contained in the Indirect Register Select register (address 0x0).

Table A. 3. Register 0x02: Interrupt and General Status.

Bit	Type	Function	Default
Bit 15	R	HALTED	0
Bit 14	R	FROZEN	0
Bit 13	R	GMODE	X
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	DBRCV_FIFO_OVRE	0
Bit 6	R	TPA_OVRE	0
Bit 5	R	TPA_UNDE	0
Bit 4	R	DBQF_SEQE	0
Bit 3	R	DBQA_SEQE	0
Bit 2	R	QA_SEQE	0
Bit 1	R	MTYPE	0
Bit 0	R	TSTAT_OVRE	0

HALTED:

This bit indicates that the EQM has responded to an assertion of the HALT input and that all EQM state machines have entered the HALTED state. HALTED is cleared on the rising edge of SYSCLK after HALT is deasserted.

FROZEN:

This bit indicates that the EQM has frozen in response to a fatal error condition. (Note: see FRZ_ON_ERROR bit in the General Control Register).

GMODE:

This bit reflects the status of the GMODE input pin.

DBRCV_FIFO_OVRE:

This error status bit indicates that the DBRCV FIFO has causing the Queue Fetch Scheduler to loose a DB. The interrupt is cleared by reading this register, but the EQM must be reset to clear the error condition.

TPA_OVRE:

This bit indicates that a TPA Overflow error (QA received while the TPA count for that SPID/P combination is 15) has occurred in 1 Gbps mode. Cleared by reading the Illegal Source Destination ID Register.

TPA_UNDE:

This bit indicates that a TPA Underflow error (DB received while the TPA count for that SPID/P combination is 0) has occurred in 1 Gbps mode. Cleared by reading the Illegal Source Destination ID Register

DBQF_SEQE:

This bit indicates that a DB-QA sequence error (DB received without first sending a QF) has occurred in 100 Mbps mode. Cleared by reading the Illegal Source Destination ID Register.

DBQA_SEQE:

This bit indicates that a DB-QA sequence error (DB received without first receiving a QA) has occurred in 100 Mbps mode. Cleared by reading the Illegal Source Destination ID Register.

QA_SEQE:

This bit indicates that a QA sequence error (QA received while the corresponding TPA bit is already set) has occurred in 100 Mbps mode. Cleared by reading the Illegal Source Destination ID Register.

MTYPE:

This error status bit indicates that a unknown or illegal message type (anything other than a QA or DB message) has been received. Cleared by reading the Illegal Message Type Register.

TSTAT_OVRE:

This error status bit indicates that the TSTAT FIFO has overflowed and some statistics have been lost. Cleared by reading this register.

Table A. 4. Register 0x03: General Control.

Bit	Type	Function	Default
Bit 15	R/W	FRZ_ON_ERROR	1
Bit 14	R/W	INIT_AFTER_HALT	1
Bit 13	R	Unused	X
Bit 12	R	Unused	X
Bit 11	R	Unused	X
Bit 10	R	Unused	X
Bit 9	R	Unused	X
Bit 8	R	Unused	X
Bit 7	R	Unused	X
Bit 6	R	Unused	X
Bit 5	R	Unused	X
Bit 4	R	Unused	X
Bit 3	R	Unused	X
Bit 2	R	Unused	X
Bit 1	R/W	SCHMODE[1]	0
Bit 0	R/W	SCHMODE[0]	0

FRZ_ON_ERROR:

When FRZ_ON_ERROR is set, the EQM will freeze after detecting a sequence error; when clear the EQM will merely ignore the out of sequence message. FRZ_ON_ERROR should be left in its default enabled state except for firmware debugging purposes.

INIT_AFTER_HALT:

When INIT_AFTER_HALT is set, the QFRP reinitialize the credit counts in the stream table. This bit should always be left set during the initial set up of the EQM. However, if the EQM is stopped during operation, this bit should be cleared before deasserting the HALT signal, in order to avoid destroying the data in the stream table.

SCHMODE[1:0]:

Determines the mode of operation for the Queue Fetch Scheduler.

Table A. 5. Scheduling Mode Control Bits.

SCHMODE	Mode	Description
00	none	QFS Disabled
01	SPS	Strict Priority Scheduling
10	WFS	Weighted Fair Scheduling
11	WPS	Weighted Priority Scheduling

Table A. 6. Indirect Register 0x00: Interrupt Enable.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R/W	DBRCV_FIFO_OVRE_INT_EN	0
Bit 6	R/W	TPA_OVRE_INT_EN	0
Bit 5	R/W	TPA_UNDE_INT_EN	0
Bit 4	R/W	DBQF_SEQE_INT_EN	0
Bit 3	R/W	DBQA_SEQE_INT_EN	0
Bit 2	R/W	QA_SEQE_INT_EN	0
Bit 1	R/W	MTYPE_INT_EN	0
Bit 0	R/W	TSTAT_OVRE_INT_EN	0

All bits: 0 = disabled, 1 = enabled

DBRCV_FIFO_OVRE_INT_EN:

This bit enables the BDRCV FIFO Overflow error interrupt.

TPA_OVRE_INT_EN:

This bit enables the TPA Overflow error interrupt.

TPA_UNDE_INT_EN:

This bit enables the TPA Underflow error interrupt.

DBQF_SEQE_INT_EN:

This bit enables the DB-QF sequence error interrupt.

DBQA_SEQE_INT_EN:

This bit enables the DB-QA sequence error interrupt.

QA_SEQE_INT_EN:

This bit enables the QA sequence error interrupt.

MTYPE_INT_EN:

This bit enables the illegal message type error interrupt.

TSTAT_OVRE_INT_EN:

This bit enables the TSTAT FIFO overflow interrupt.

Table A. 7. Indirect Register 0x01: Base Address Offset 1.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R/W	TXST_BASE[11]	0
Bit 10	R/W	TXST_BASE[10]	0
Bit 9	R/W	TXST_BASE[9]	0
Bit 8	R/W	TXST_BASE[8]	0
Bit 7	R/W	TXST_BASE[7]	0
Bit 6	R/W	TXST_BASE[6]	0
Bit 5	R/W	TXST_BASE[5]	0
Bit 4	R/W	TXST_BASE[4]	0
Bit 3	R/W	TXST_BASE[3]	0
Bit 2	R/W	TXST_BASE[2]	0
Bit 1	R/W	TXST_BASE[1]	0
Bit 0	R/W	TXST_BASE[0]	0

TXST_BASE[11:0]:

The TXST_BASE sets the base address of the Transmit Statistics Array stored in external SSRAM.

The 22-bit DWord address is calculated as follows:

address = {TXST_BASE[11:0], 0000000000}
+ {00000,TXSTAT[15:0], TXST_Dword}

Table A. 8. Indirect Register 0x02: Illegal Source/Destination Field.

Bit	Type	Function	Default
Bit 15	R	DCID[5]	0
Bit 14	R	DCID[4]	0
Bit 13	R	DCID[3]	0
Bit 12	R	DCID[2]	0
Bit 11	R	DCID[1]	0
Bit 10	R	DCID[0]	0
Bit 9	R	SPID[9]	0
Bit 8	R	SPID[8]	0
Bit 7	R	SPID[7]	0
Bit 6	R	SPID[6]	0
Bit 5	R	SPID[5]	0
Bit 4	R	SPID[4]	0
Bit 3	R	SPID[3]	0
Bit 2	R	SPID[2]	0
Bit 1	R	SPID[1]	0
Bit 0	R	SPID[0]	0

SPID[9:0]:

The SPID field reports the SPID field of a QA or DB message received out of sequence.

DCID[5:0]:

The DCID field reports the DCID field of a QA or DB message received out of sequence.

Table A. 9. Indirect Register 0x03: Illegal Message Type Field.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R	Unused	0
Bit 4	R	Unused	0
Bit 3	R	TYPE[3]	0
Bit 2	R	TYPE[2]	0
Bit 1	R	TYPE[1]	0
Bit 0	R	TYPE[0]	0

TYPE[3:0]:

The TYPE field reports the TYPE field of an unrecognized message received by the QADBP.

Table A. 10. Indirect Register 0x04: Global Maximum QF Outstanding.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R/W	GMAX_QF[7]	0
Bit 6	R/W	GMAX_QF[6]	0
Bit 5	R/W	GMAX_QF[5]	0
Bit 4	R/W	GMAX_QF[4]	0
Bit 3	R/W	GMAX_QF[3]	1
Bit 2	R/W	GMAX_QF[2]	0
Bit 1	R/W	GMAX_QF[1]	1
Bit 0	R/W	GMAX_QF[0]	0

GMAX_QF[7:0]:

The GMAX_QF field controls the maximum number of QFs that the QFS may issue across all traffic streams in 1 Gbps mode. A nominal value of 10 is recommended, although the value may be set as high as the maximum number of memory blocks in the TXCTRL Frame Reassembly Buffer (i.e., 70 for block size of 120 bytes, 140 for block size of 240 bytes). Setting GMAX_QF larger than the number of blocks in the TXCTRL FRB will not permit more QFs to be issued; however, it may impair the scheduling algorithm.

This setting has no effect in 100 Mbps mode.

Table A. 11. Indirect Register 0x05: Maximum QF Outstanding per Stream.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R	Unused	0
Bit 4	R	Unused	0
Bit 3	R/W	SMAX_QF[3]	1
Bit 2	R/W	SMAX_QF[2]	0
Bit 1	R/W	SMAX_QF[1]	1
Bit 0	R/W	SMAX_QF[0]	0

SMAX_QF[7:0]:

The SMAX_QF field controls the maximum number of QFs that the QFS may issue for any given traffic stream in 1 Gbps mode. This setting has no effect in 100 Mbps mode. A nominal value of 10 is recommended.

Table A. 12. Indirect Register 0x06, 0x09, 0x0C, 0x0F: Channel Weighted Scheduling (0-3) Limit.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R/W	WSn_LIMIT[7]	0
Bit 6	R/W	WSn_LIMIT[6]	0
Bit 5	R/W	WSn_LIMIT[5]	0
Bit 4	R/W	WSn_LIMIT[4]	0
Bit 3	R/W	WSn_LIMIT[3]	0
Bit 2	R/W	WSn_LIMIT[2]	0
Bit 1	R/W	WSn_LIMIT[1]	0
Bit 0	R/W	WSn_LIMIT[0]	0

WSn_LIMIT[7:0]:

The WSn_LIMIT sets the limit value for the channel Weighted Scheduling Credit Store. There are four WS limit control registers (one for each class of service) for each channel. In 1 Gbps mode, only channel 0 is used.

Note that there is one set of weighted scheduling control registers per channel and the set is selected by the CHAN_SEL[2:0] pins.

Table A. 13. Indirect Register 0x07, 0x0A, 0x0D, 0x10: Channel Weighted Scheduling (0-3) Weight.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R/W	WSn_WEIGHT[7]	0
Bit 6	R/W	WSn_WEIGHT[6]	0
Bit 5	R/W	WSn_WEIGHT[5]	0
Bit 4	R/W	WSn_WEIGHT[4]	0
Bit 3	R/W	WSn_WEIGHT[3]	0
Bit 2	R/W	WSn_WEIGHT[2]	0
Bit 1	R/W	WSn_WEIGHT[1]	0
Bit 0	R/W	WSn_WEIGHT[0]	0

WSn_WEIGHT[7:0]:

The WSn_WEIGHT sets the weight value for the channel weighted scheduling counters. There are four WS weight control registers (one for each class of service) for each channel. In 1 Gbps mode, only channel 0 is used.

Note that there is one set of weighted scheduling control registers per channel and the set is selected by the CHAN_SEL[2:0] pins.

Table A. 14. Indirect Register 0x08, 0x0B, 0x0E, 0x11: Channel Weighted Scheduling (0-3) Credit Store.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	WSn_CREDITS[7]	0
Bit 6	R	WSn_CREDITS[6]	0
Bit 5	R	WSn_CREDITS[5]	0
Bit 4	R	WSn_CREDITS[4]	0
Bit 3	R	WSn_CREDITS[3]	0
Bit 2	R	WSn_CREDITS[2]	0
Bit 1	R	WSn_CREDITS[1]	0
Bit 0	R	WSn_CREDITS[0]	0

WSn_CREDITS[7:0]:

The WSn_CREDITS contains the running count value for the channel weighted scheduling counters.

There are four WS Credit Store status registers (one for each class of service) for each channel. In 1 Gbps mode, only channel 0 is used.

Note that there is one set of weighted scheduling control registers per channel and the set is selected by the CHAN_SEL[2:0] pins.

Table A. 15. Indirect Register 0x12: Weighted Scheduling Threshold.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R/W	WS_THRESH[5]	0
Bit 4	R/W	WS_THRESH[4]	0
Bit 3	R/W	WS_THRESH[3]	0
Bit 2	R/W	WS_THRESH[2]	1
Bit 1	R/W	WS_THRESH[1]	1
Bit 0	R/W	WS_THRESH[0]	1

WS_THRESH[5:0]:

The WS_THRESH contains the number of DB payloads required to assemble a maximum sized Ethernet frame. Divide 1518 (1522 if tag insertion is in use) by the DB maximum payload size and round up to the next integer value to determine WS_THRESH (see Table 9 for examples).

Table A. 16. Weighted Scheduling Threshold Values.

DB Maximum Payload Size	WS_THRESH
120	13
180	9
240	7

Table A. 17. Indirect Register 0x20: TPA Diagnostic RAM Control Register.

Bit	Type	Function	Default
Bit 15	R	TPA_DONE	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R/W	TPA_RAM_NUM[2]	0
Bit 6	R/W	TPA_RAM_NUM[1]	0
Bit 5	R/W	TPA_RAM_NUM[0]	0
Bit 4	R/W	TPA_ADDR [4]	0
Bit 3	R/W	TPA_ADDR [3]	0
Bit 2	R/W	TPA_ADDR [2]	0
Bit 1	R/W	TPA_ADDR [1]	0
Bit 0	R/W	TPA_ADDR[0]	0

Writing to the TPA Diagnostic RAM Control register will trigger a read of the indicated TPA RAM. The RAM data will be latched into the TPA Diagnostic Data registers and the TPA_DONE bit asserted after 4 clock cycles.

TPA_RAM_DONE:

TPA_DONE indicates that the TPA RAM read has been completed. TPA_DONE is cleared by reading the TPA Diagnostic Data High Register.

TPA_RAM_NUM[4:0]:

TPA_RAM_NUM contains the TPA RAM number (0-7) for diagnostic address. In 100 Mbps mode, the RAM number corresponds to the channel number. In 1 Gbps mode, RAM 0 contains the LSB and RAM 3 the MSB of the 4-bit TPA value.

TPA_ADDR[4:0]:

TPA_ADDR contains the TPA RAM address for diagnostic address.

Table A. 18. Indirect Register 0x21: TPA Diagnostic Data High Register.

Bit	Type	Function	Default
Bit 15	R	TPA_DATA[31]	0
Bit 14	R	TPA_DATA[30]	0
Bit 13	R	TPA_DATA[29]	0
Bit 12	R	TPA_DATA[28]	0
Bit 11	R	TPA_DATA[27]	0
Bit 10	R	TPA_DATA[26]	0
Bit 9	R	TPA_DATA[25]	0
Bit 8	R	TPA_DATA[24]	0
Bit 7	R	TPA_DATA[23]	0
Bit 6	R	TPA_DATA[22]	0
Bit 5	R	TPA_DATA[21]	0
Bit 4	R	TPA_DATA[20]	0
Bit 3	R	TPA_ DATA [19]	0
Bit 2	R	TPA_ DATA [18]	0
Bit 1	R	TPA_ DATA [17]	0
Bit 0	R	TPA_ DATA [16]	0

TPA_DATA[31:16]:

TPA_DATA contains the most significant word of data latched from the TPA RAM during a diagnostic access. The access is triggered by writing to the TPA Diagnostic RAM Control register.

Table A. 19. Indirect Register 0x22: TPA Diagnostic Data Low Register.

Bit	Type	Function	Default
Bit 15	R	TPA_DATA[15]	0
Bit 14	R	TPA_DATA[14]	0
Bit 13	R	TPA_DATA[13]	0
Bit 12	R	TPA_DATA[12]	0
Bit 11	R	TPA_DATA[11]	0
Bit 10	R	TPA_DATA[10]	0
Bit 9	R	TPA_DATA[9]	0
Bit 8	R	TPA_DATA[8]	0
Bit 7	R	TPA_DATA[7]	0
Bit 6	R	TPA_DATA[6]	0
Bit 5	R	TPA_DATA[5]	0
Bit 4	R	TPA_DATA[4]	0
Bit 3	R	TPA_ DATA [3]	0
Bit 2	R	TPA_ DATA [2]	0
Bit 1	R	TPA_ DATA [1]	0
Bit 0	R	TPA_ DATA [0]	0

TPA_DATA[15:0]:

TPA_DATA contains the least significant word of data latched from the TPA RAM during a diagnostic access. The access is triggered by writing to the TPA Diagnostic RAM Control register.

Table A. 20. Indirect Register 0x23: QFS Stream Table Index Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R	Unused	0
Bit 4	R	Unused	0
Bit 3	R/W	DIAG_ST_INDEX [3]	0
Bit 2	R/W	DIAG_ST_INDEX [2]	0
Bit 1	R/W	DIAG_ST_INDEX [1]	0
Bit 0	R/W	DIAG_ST_INDEX [0]	0

DIAG_ST_INDEX[15:0]:

DIAG_ST_INDEX controls which stream table entry (0 through 9) table will be available in registers 0x24 and 0x25. Caution: writing this register doesnot latch the stream table data, it merely controls a multiplexer on the output of the table. Therefore subsequent reads of registers 0x24 and 0x25 may differ.

Note: This register is specific to the 1 Gbps mode of operation

Table A. 21. Indirect Register 0x24: QFS Stream Table SPID/P Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	DIAG_ST_ACTIVE	0
Bit 9	R	DIAG_ST_P[1]	0
Bit 8	R	DIAG_ST_P[0]	0
Bit 7	R	DIAG_ST_SPID[7]	0
Bit 6	R	DIAG_ST_ SPID[6]	0
Bit 5	R	DIAG_ST_ SPID[5]	0
Bit 4	R	DIAG_ST_ SPID[4]	0
Bit 3	R	DIAG_ST_SPID[3]	0
Bit 2	R	DIAG_ST_ SPID[2]	0
Bit 1	R	DIAG_ST_ SPID[1]	0
Bit 0	R	DIAG_ST_ SPID[0]	0

DIAG_ST_ACTIVE:

DIAG_ST_ACTIVE indicates if the specified stream is active (1) or inactive (0).

DIAG_ST_P [1:0]:

DIAG_ST_P contains the priority level assigned to the stream specified in the Diagnostic Stream Table Index Register.

DIAG_ST_SPID [7:0]:

DIAG_ST_P contains the source port ID assigned to the stream specified in the Diagnostic Stream Table Index Register.

Table A. 22. Indirect Register 0x25: QFS Stream QF/Credit Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	ST_CREDITS[3]	0
Bit 10	R	ST_CREDITS[2]	0
Bit 9	R	ST_CREDITS[1]	0
Bit 8	R	ST_CREDITS[0]	0
Bit 7	R	QF_PENDING[3]	0
Bit 6	R	QF_PENDING[2]	0
Bit 5	R	QF_PENDING[1]	0
Bit 4	R	QF_PENDING[0]	0
Bit 3	R	QF_OUTSTANDING[3]	0
Bit 2	R	QF_OUTSTANDING[2]	0
Bit 1	R	QF_OUTSTANDING[1]	0
Bit 0	R	QF_OUTSTANDING[0]	0

ST_CREDITS[3]:

ST_CREDITS indicates the number of DB scheduling credits reserved for the stream specified in the Diagnostic Stream Table Index Register.

QF_PENDING[3:0]:

QF_PENDING indicates the number of QF requests accepted from the TPA Scanner but which have not yet been transmitted for the stream specified in the Diagnostic Stream Table Index Register.

QF_OUTSTANDING[3:0]:

QF_OUTSTANDING indicates the number of QFs which have been transmitted, but for which a DB with EOF set has not been returned for the stream specified in the Diagnostic Stream Table Index Register.

Table A. 23. Indirect Register 0x26: QFS Channel Select Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	Unused	0
Bit 6	R	Unused	0
Bit 5	R	Unused	0
Bit 4	R	Unused	0
Bit 3	R	Unused	0
Bit 2	R/W	DIAG_CHAN_SEL[2]	0
Bit 1	R/W	DIAG_CHAN_SEL[1]	0
Bit 0	R/W	DIAG_CHAN_SEL[0]	0

DIAG_CHAN_SEL[2:0]:

DIAG_CHAN_SEL controls which 100 Mbps (0 through 7) channel's state information will be available in The diagnostic channel state register. Note: This register is specific to the 100 Mbps mode of operation.

Table A. 24. Indirect Register 0x27: QFS Channel Status Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	DIAG_SCAN_STATE[3]	0
Bit 12	R	DIAG_SCAN_STATE[2]	0
Bit 11	R	DIAG_SCAN_STATE[1]	0
Bit 10	R	DIAG_SCAN_STATE[0]	0
Bit 9	R	DIAG_QF_P[1]	0
Bit 8	R	DIAG_QF_P[0]	0
Bit 7	R	DIAG_QF_SPID[7]	0
Bit 6	R	DIAG_QF_SPID[6]	0
Bit 5	R	DIAG_QF_SPID[5]	0
Bit 4	R	DIAG_QF_SPID[4]	0
Bit 3	R	DIAG_QF_SPID[3]	0
Bit 2	R	DIAG_QF_SPID[2]	0
Bit 1	R	DIAG_QF_SPID[1]	0
Bit 0	R	DIAG_QF_SPID[0]	0

DIAG_SCAN_STATE[3:0]:

DIAG_SCAN_STATE indicates the internal state of the TPA Scanner. This value can be used to determine if the TPA Scanner has become locked up due to a lost or corrupted message. If this register is read twice and the value changes, the state machine is not locked up. A consistent value of 0xC means the scanner is waiting for access to the EXI bus. A consistent value of 0xD indicates the scanner has sent a QF and is now waiting for a DB in return. A consistent value of 0xF means the state machine is halted due to an assertion of the HALT input. The state machine can not hold in other states. Note: This register is specific to the 100 Mbps mode of operation.

DIAG_QF_P [1:0]:

DIAG_QF_P contains the priority level of the last QF sent for this channel.

DIAG_QF_SPID [7:0]:

DIAG_QF_SPID contains the source port ID of the last QF sent for this channel.

Table A. 25. Indirect Register 0x28: QFS TXD_BLK_AVAIL Register.

Bit	Type	Function	Default
Bit 15	R	Unused	0
Bit 14	R	Unused	0
Bit 13	R	Unused	0
Bit 12	R	Unused	0
Bit 11	R	Unused	0
Bit 10	R	Unused	0
Bit 9	R	Unused	0
Bit 8	R	Unused	0
Bit 7	R	TXD_BLK_AVAIL[7]	0
Bit 6	R	TXD_BLK_AVAIL[6]	0
Bit 5	R	TXD_BLK_AVAIL[5]	0
Bit 4	R	TXD_BLK_AVAIL[4]	0
Bit 3	R	TXD_BLK_AVAIL[3]	0
Bit 2	R	TXD_BLK_AVAIL[2]	0
Bit 1	R	TXD_BLK_AVAIL[1]	0
Bit 0	R	TXD_BLK_AVAIL[0]	0

TXD_BLK_AVAIL[7:0]:

TXD_BLK_AVAIL indicates the state of the TXD_BLK_AVAIL input to the EQM. In 1 Gbps mode, TXD_BLK_AVAIL indicates the number of free buffers in the downstream frame reassembly buffer. In 100 Mbps mode, each bit of TXD_BLK_AVAIL acts as a flow control bit for the corresponding channel; i.e., TXD_BLK_AVAIL[n] = 1 indicates that the downstream frame reassembly FIFO is ready to accept further DBs from channel n.

Appendix B: EQM Test Register Descriptions

Test mode registers are used to apply test vectors during production testing of the EQM. Test mode registers (as opposed to normal mode registers) are selected when TRSB = 0. The two supported test modes are logic scan and RAM Built In Self Test (BIST) scan.

Writing values into unused register bits has no effect. Reading unused bits can produce either a logic 1 or a logic 0; therefore, unused register bits should be masked off by software when read.

Writeable test mode register bits are not initialized upon reset unless otherwise noted.

Test registers are write only; consequently, the value written cannot be read back. The value returned on CRRDATA[15:0] depends only on the mode selected (SCAN or BIST) not on the address used.

Table B. 1. Test Register 1: Test Mode Select.

Bit	Type	Function	Default
Bit 15	W	GMODE_TEST	X
Bit 14	W	Unused	X
Bit 13	W	Unused	X
Bit 12	W	Unused	X
Bit 11	W	Unused	X
Bit 10	W	Unused	X
Bit 9	W	Unused	X
Bit 8	W	Unused	X
Bit 7	W	Unused	X
Bit 6	W	Unused	X
Bit 5	W	Unused	X
Bit 4	W	Unused	X
Bit 3	W	Unused	X
Bit 2	W	Unused	X
Bit 1	W	TMS[1]	X
Bit 0	W	TMS[0]	X

GMODE_TEST:

Allows control of the static GMODE input for test purposes. When TSTB=1, the GMODE_TEST register output is multiplexed onto the internal GMODE signal, replacing the external GMODE signal

TMS [1:0]:

Test mode select: Select one of 4 test modes as shown in the table below:

Table B. 2. Test Modes.

TMS[1]	TMS[0]	Mode	Description	CRRDATA
0	0	TM0	Unused	0x0000
0	1	TM1	Core logic (SCAN)	SCANOUT[15:0]
1	0	TM2	RAM (BIST)	BISTOUT[15:0]

TMS[1]	TMS[0]	Mode	Description	CRRDATA
1	1	TM3	Unused	0x0000

B.1 Test Mode 1: Full scan test logic

REGISTER DESCRIPTION:

Table B. 3. Test Register 0x00: Test Enable.

Bit	Type	Function	Default
Bit 15	W	Unused	X
Bit 14	W	Unused	X
Bit 13	W	Unused	X
Bit 12	W	Unused	X
Bit 11	W	Unused	X
Bit 10	W	Unused	X
Bit 9	W	Unused	X
Bit 8	W	Unused	X
Bit 7	W	Unused	X
Bit 6	W	Unused	X
Bit 5	W	Unused	X
Bit 4	W	Unused	X
Bit 3	W	Unused	X
Bit 2	W	Unused	X
Bit 1	W	Unused	X
Bit 0	W	SCAN_EN	X

SCAN_EN:

Enable shifting of the scan registers.

Table B. 4. Test Register 0x02: Scan Test Data Input.

Bit	Type	Function	Default
Bit 15	W	SCAN_IN [15]	X
Bit 14	W	SCAN_IN [14]	X
Bit 13	W	SCAN_IN [13]	X
Bit 12	W	SCAN_IN [12]	X
Bit 11	W	SCAN_IN [11]	X
Bit 10	W	SCAN_IN [10]	X
Bit 9	W	SCAN_IN [9]	X
Bit 8	W	SCAN_IN [8]	X
Bit 7	W	SCAN_IN [7]	X
Bit 6	W	SCAN_IN [6]	X
Bit 5	W	SCAN_IN [5]	X
Bit 4	W	SCAN_IN [4]	X
Bit 3	W	SCAN_IN [3]	X
Bit 2	W	SCAN_IN [2]	X
Bit 1	W	SCAN_IN [1]	X
Bit 0	W	SCAN_IN [0]	X

SCAN_IN[15:0]:

Scan chain input data.

Table B. 5. SCAN Mode Data Output.

Bit	Type	Function	Default
Bit 15	R	SCAN_OUT [15]	X
Bit 14	R	SCAN_OUT [14]	X
Bit 13	R	SCAN_OUT [13]	X
Bit 12	R	SCAN_OUT [12]	X
Bit 11	R	SCAN_OUT [11]	X
Bit 10	R	SCAN_OUT [10]	X
Bit 9	R	SCAN_OUT [9]	X
Bit 8	R	SCAN_OUT [8]	X
Bit 7	R	SCAN_OUT [7]	X
Bit 6	R	SCAN_OUT [6]	X
Bit 5	R	SCAN_OUT [5]	X
Bit 4	R	SCAN_OUT [4]	X
Bit 3	R	SCAN_OUT [3]	X
Bit 2	R	SCAN_OUT [2]	X
Bit 1	R	SCAN_OUT [1]	X
Bit 0	R	SCAN_OUT [0]	X

SCAN_OUT[15:0]:

Scan chain output data. The logic scan test is carried out by initializing the test mode appropriately (the using test mode select register) and then repetitively writing a scan pattern into test register 2, pulsing SYSCLK to advance the scan chains and then reading the resulting scan pattern out of test register 2. Since SYSCLK is used to clock the scan chains (as in normal logic operation), CRTCLK is used to clock the test register logic.

B.2 Test Mode 2: RAM BIST

Table B. 6. Test Register 0x00: Test Enable.

Bit	Type	Write Function	Read Function
Bit 15	W	Unused	X
Bit 14	W	Unused	X
Bit 13	W	Unused	X
Bit 12	W	Unused	X
Bit 11	W	Unused	X
Bit 10	W	Unused	X
Bit 9	W	Unused	X
Bit 8	W	Unused	X
Bit 7	W	Unused	X
Bit 6	W	Unused	X
Bit 5	W	Unused	X
Bit 4	W	Unused	X
Bit 3	W	Unused	X
Bit 2	W	Unused	X
Bit 1	W	Unused	X
Bit 0	W	BIST_EN	X

BIST_EN:

Enable the internal RAM BIST.

Table B. 7. Test Register 0x02: BIST Test Data.

Bit	Type	Function	Default
Bit 15	W	Unused	X
Bit 14	W	Unused	X
Bit 13	W	Unused	X
Bit 12	W	Unused	X
Bit 11	W	Unused	X
Bit 10	W	Unused	X
Bit 9	W	Unused	X
Bit 8	W	Unused	X
Bit 7	W	BIST_DATA[7]	X
Bit 6	W	BIST_DATA[6]	X
Bit 5	W	BIST_DATA[5]	X
Bit 4	W	BIST_DATA[4]	X
Bit 3	W	BIST_DATA[3]	X
Bit 2	W	BIST_DATA[2]	X
Bit 1	W	BIST_DATA[1]	X
Bit 0	W	BIST_DATA[0]	X

BIST_DATA[7:0]:

Seed data for the BIST sequence.

Table B. 8. BIST Mode Data Output.

Bit	Type	Function	Default
Bit 15	R	Unused	X
Bit 14	R	Unused	X
Bit 13	R	Unused	X
Bit 12	R	Unused	X
Bit 11	R	Unused	X
Bit 10	R	BIST_ERROR[8]	X
Bit 9	R	BIST_ERROR[7]	X
Bit 8	R	BIST_ERROR[6]	X
Bit 7	R	BIST_ERROR[5]	X
Bit 6	R	BIST_ERROR[4]	X
Bit 5	R	BIST_ERROR[3]	X
Bit 4	R	BIST_ERROR[2]	X
Bit 3	R	BIST_ERROR[1]	X
Bit 2	R	BIST_ERROR[0]	X
Bit 1	R	BIST_RESULT	X
Bit 0	R	BIST_END	X

BIST_ERROR[3:0]:

On each clock cycle : high if a data write-read-verify comparison mismatch is detected for the associated RAM cell : low otherwise. The associated RAMs are shown in the table following.

The address of the RAM location giving the error may be deduced from the clock cycle in which the error is detected relative to the start of the BIST pass as noted below.

Table B. 9. BIST_ERROR Bits.

Bit	RAM
8	TSTAT FIFO RAM
7	TPA RAM 7
6	TPA RAM 6
5	TPA RAM 5
4	TPA RAM 4
3	TPA RAM 3
2	TPA RAM 2
1	TPA RAM 1
0	TPA RAM 0

BIST_RESULT:

After the BIST sequence completes, this bit indicates the overall result. If any errors were detected at any time, this bit will be 0. Otherwise, 1 indicates that the RAM BIST was successful. This bit is only valid when BIST_END is asserted high.

BIST_END:

Set high at the end of the BIST sequence.

The RAM BIST test is carried out by initializing the test mode appropriately, repetitively pulsing SYSCLK to run the BIST sequence and finally reading test register 2 to obtain the result. If the BIST fails, rerunning the sequence and monitoring the BIST_ERROR bits in test register 2 on every SYSCLK cycle will reveal the RAM location which is failing. The nine RAMs are tested in parallel; therefore, the test time is determined by the deepest RAM, dpr32x32. The RAM BIST sequence involves 16 accesses to each memory location : thus, the RAM BIST takes approximately 1032 clocks.

Appendix C: Performance Simulation Details

The results of the weighted scheduling performance simulations demonstrate the effectiveness of the scheduling algorithms under different operating modes and with a variety of weighting parameters and operating conditions. Table C.1 is a key for the simulation results, which are presented in Tables C.2 through C.5.

Table C. 1. Performance Tests Parameter Key.

Parameter	Key	Description
Chip	1x1G	PM3380 1 x 1 Gbps port controller
	8x100M	PM3370 8 x 100 Mbps port controller
Scheduling Mode	WPS	Weighted Priority Scheduling
	WFS	Weighted Fair Scheduling
Channels	1	Single channel
	8	Eight channels (used only with FELIX chip)
Streams	2/4/10/12/32	Number of unique Source Port/Priority streams in service.
Packet Size	T	Tinygrams – smallest Ethernet packet (64 bytes)
	W	Maxigrams – Largest Ethernet packet size (1822 bytes) including VLAN tag
	DB	All packets are multiples of the default DB payload size (i.e., n x 240 bytes; n = 1 to 7)
	R	Packets are randomly sized between 64 and 1522 bytes
Weights	4/8/16/32	The weighting credits applied to the four classes of service (or priority levels). Even weights of two different values were checked to show that traffic could be scheduled evenly across all classes of service. An arithmetic ratio of weights was checked to show situations where the worst ration between two classes of service is 4:1. Two geometric ratios of weights were checked to show situations where the worst ration between two classes of service is 16:1. Finally a corner case where one class of service gets a minimum weighting, a second class gets maximum weighting and the others classes are unprovisioned is used to check worst case conditions.
	16/32/64/128	
	7/7/7/7	
	25/25/25/25	
	15/30/45/60	
	1/128/0/0	
Limit : Weight Ratio	1 : 1 2 : 1 3 : 1 4 : 1	The ration between the limit and weight parameters for credits affects the “memory” of the scheduler; i.e., how many unused credits can be retained by a given class of service after each service round.

Table C. 2. WFS Performance Tests – 1 Gbps mode.

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
1x1G	WFS	1	4	DB	4/8/16/32	2	512	0	6.7%	3.6%	-3.1%
								1	13.3%	14.3%	1.0%
								2	26.7%	28.0%	1.3%
								3	53.3%	54.2%	0.9%
1x1G	WFS	1	4	DB	4/8/16/32	3	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.3%	-1.0%
1x1G	WFS	1	4	DB	4/8/16/32	4	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.3%	-1.0%
1x1G	WFS	1	10	DB	4/8/16/32	2	512	0	6.7%	5.9%	-0.8%
								1	13.3%	14.1%	0.8%
								2	26.7%	27.2%	0.5%
								3	53.3%	52.7%	-0.6%
1x1G	WFS	1	12	DB	4/8/16/32	2	512	0	6.7%	5.3%	-1.4%
								1	13.3%	14.0%	0.7%
								2	26.7%	27.5%	0.8%
								3	53.3%	53.3%	0.0%
1x1G	WFS	1	4	DB	16/32/64/128	1	512	0	6.7%	4.8%	-1.9%
								1	13.3%	12.5%	-0.8%
								2	26.7%	28.3%	1.6%
								3	53.3%	54.4%	1.1%
1x1G	WFS	1	4	DB	16/32/64/128	2	512	0	6.7%	6.7%	0.0%
								1	13.3%	13.7%	0.4%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.5%	-0.8%
1x1G	WFS	1	10	DB	16/32/64/128	2	512	0	6.7%	7.0%	0.3%
								1	13.3%	14.0%	0.7%
								2	26.7%	26.9%	0.2%
								3	53.3%	52.1%	-1.2%
1x1G	WFS	1	12	DB	16/32/64/128	2	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.7%	0.4%
								2	26.7%	27.0%	0.3%
								3	53.3%	52.5%	-0.8%
1x1G	WFS	1	32	R	16/32/64/128	2	368	0	6.7%	6.4%	-0.3%
								1	13.3%	13.2%	-0.1%
								2	26.7%	26.8%	0.1%
								3	53.3%	53.6%	0.3%
1x1G	WFS	1	4	DB	7/7/7/7	3	512	0	25.0%	25.2%	0.2%
								1	25.0%	24.8%	-0.2%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
1x1G	WFS	1	32	DB	7/7/7/7	3	512	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
1x1G	WFS	1	4	DB	25/25/25/25	2	512	0	25.0%	24.3%	-0.7%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.4%	0.4%
								3	25.0%	25.3%	0.3%
1x1G	WFS	1	32	M	25/25/25/25	2	512	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
1x1G	WFS	1	4	T	15/30/45/60	4	512	0	10.0%	9.9%	-0.1%
								1	20.0%	20.3%	0.3%
								2	30.0%	30.6%	0.6%
								3	40.0%	39.2%	-0.8%
1x1G	WFS	1	12	T	15/30/45/60	4	512	0	10.0%	10.1%	0.1%
								1	20.0%	20.3%	0.3%
								2	30.0%	30.2%	0.2%
								3	40.0%	39.5%	-0.5%

Table C. 3. WPS Performance Tests – 1 Gbps mode.

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
1x1G	WPS	1	4	DB	4/8/16/32	2	512	0	6.7%	3.6%	-3.1%
								1	13.3%	14.3%	1.0%
								2	26.7%	28.0%	1.3%
								3	53.3%	54.2%	0.9%
1x1G	WPS	1	4	DB	4/8/16/32	3	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.3%	-1.0%
1x1G	WPS	1	4	DB	4/8/16/32	4	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.3%	-1.0%
1x1G	WPS	1	10	DB	4/8/16/32	2	512	0	6.7%	5.9%	-0.8%
								1	13.3%	14.1%	0.8%
								2	26.7%	27.2%	0.5%
								3	53.3%	52.7%	-0.6%
1x1G	WPS	1	12	DB	4/8/16/32	2	512	0	6.7%	5.3%	-1.4%
								1	13.3%	14.0%	0.7%
								2	26.7%	27.5%	0.8%
								3	53.3%	53.3%	0.0%
1x1G	WPS	1	4	DB	16/32/64/128	1	512	0	6.7%	4.8%	-1.9%
								1	13.3%	12.5%	-0.8%
								2	26.7%	28.3%	1.6%
								3	53.3%	54.4%	1.1%
1x1G	WPS	1	4	DB	16/32/64/128	2	512	0	6.7%	6.7%	0.0%
								1	13.3%	13.7%	0.4%
								2	26.7%	27.1%	0.4%
								3	53.3%	52.5%	-0.8%
1x1G	WPS	1	10	DB	16/32/64/128	2	512	0	6.7%	7.0%	0.3%
								1	13.3%	14.0%	0.7%
								2	26.7%	26.9%	0.2%
								3	53.3%	52.1%	-1.2%
1x1G	WPS	1	12	DB	16/32/64/128	2	512	0	6.7%	6.8%	0.1%
								1	13.3%	13.7%	0.4%
								2	26.7%	27.0%	0.3%
								3	53.3%	52.5%	-0.8%
1x1G	WPS	1	32	R	16/32/64/128	2	368	0	6.7%	6.6%	-0.1%
								1	13.3%	13.3%	0.0%
								2	26.7%	26.8%	0.1%
								3	53.3%	53.4%	0.1%
1x1G	WPS	1	4	DB	7/7/7/7	3	512	0	25.0%	25.2%	0.2%
								1	25.0%	24.8%	-0.2%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
1x1G	WPS	1	32	M	7/7/7/7	3	512	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
1x1G	WPS	1	4	DB	25/25/25/25	2	512	0	25.0%	24.3%	-0.7%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.4%	0.4%
								3	25.0%	25.3%	0.3%
1x1G	WPS	1	32	M	25/25/25/25	2	512	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
1x1G	WPS	1	4	T	15/30/45/60	4	512	0	10.0%	10.6%	0.6%
								1	20.0%	21.1%	1.1%
								2	30.0%	31.0%	1.0%
								3	40.0%	37.4%	**-2.6%**
1x1G	WPS	1	12	T	15/30/45/60	4	512	0	10.0%	10.1%	0.1%
								1	20.0%	20.2%	0.2%
								2	30.0%	30.2%	0.2%
								3	40.0%	39.4%	-0.6%

Table C. 4. WFS Performance Tests – 100 Mbps mode.

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
8x100M	WFS	1	4	DB	4/8/16/32	2	256	0	6.7%	3.4%	-3.3%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.9%	1.2%
								3	53.3%	54.9%	1.6%
8x100M	WFS	1	4	DB	4/8/16/32	3	256	0	6.7%	6.4%	-0.3%
								1	13.3%	13.4%	0.1%
								2	26.7%	27.0%	0.3%
								3	53.3%	53.2%	-0.1%
8x100M	WFS	1	4	DB	4/8/16/32	4	256	0	6.7%	6.4%	-0.3%
								1	13.3%	13.4%	0.1%
								2	26.7%	27.0%	0.3%
								3	53.3%	53.2%	-0.1%
8x100M	WFS	1	12	DB	4/8/16/32	2	256	0	6.7%	5.0%	-1.7%
								1	13.3%	13.5%	0.2%
								2	26.7%	27.3%	0.6%
								3	53.3%	54.2%	0.9%
8x100M	WFS	1	4	DB	16/32/64/128	1	256	0	6.7%	4.3%	-2.4%
								1	13.3%	12.2%	-1.1%
								2	26.7%	28.1%	1.4%
								3	53.3%	55.4%	2.1%
8x100M	WFS	1	4	DB	16/32/64/128	2	256	0	6.7%	6.3%	-0.4%
								1	13.3%	13.9%	0.6%
								2	26.7%	28.0%	1.3%
								3	53.3%	51.9%	-1.4%
8x100M	WFS	1	12	DB	16/32/64/128	2	256	0	6.7%	6.5%	-0.2%
								1	13.3%	13.2%	-0.1%
								2	26.7%	27.5%	0.8%
								3	53.3%	52.9%	-0.4%
8x100M	WFS	1	4	M	7/7/7/7	3	256	0	25.0%	25.1%	0.1%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WFS	1	4	DB	7/7/7/7	3	256	0	25.0%	24.9%	-0.1%
								1	25.0%	24.8%	-0.2%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.2%	0.2%
8x100M	WFS	1	32	T	7/7/7/7	3	256	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WFS	1	4	DB	25/25/25/25	2	256	0	25.0%	25.8%	0.8%
								1	25.0%	24.7%	-0.3%
								2	25.0%	24.7%	-0.3%
								3	25.0%	24.9%	-0.1%

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
8x100M	WFS	1	32	T	25/25/25/25	2	256	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WFS	1	4	T	15/30/45/60	4	256	0	10.0%	8.0%	-2.0%
								1	20.0%	21.4%	1.4%
								2	30.0%	32.6%	2.6%
								3	40.0%	37.9%	-2.1%
8x100M	WFS	1	12	T	15/30/45/60	4	256	0	10.0%	9.1%	-0.9%
								1	20.0%	20.2%	0.2%
								2	30.0%	30.4%	0.4%
								3	40.0%	40.3%	0.3%

Table C. 5. WPS Performance Tests – 100 Mbps mode.

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
8x100M	WPS	1	4	DB	4/8/16/32	2	256	0	6.7%	3.4%	-3.3%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.9%	1.2%
								3	53.3%	54.9%	1.6%
8x100M	WPS	1	4	DB	4/8/16/32	3	256	0	6.7%	6.6%	-0.1%
								1	13.3%	13.4%	0.1%
								2	26.7%	27.0%	0.3%
								3	53.3%	53.1%	-0.2%
8x100M	WPS	1	4	DB	4/8/16/32	4	256	0	6.7%	6.6%	-0.1%
								1	13.3%	13.4%	0.1%
								2	26.7%	27.0%	0.3%
								3	53.3%	53.1%	-0.2%
8x100M	WPS	1	12	DB	4/8/16/32	2	256	0	6.7%	5.1%	-1.6%
								1	13.3%	13.6%	0.3%
								2	26.7%	27.3%	0.6%
								3	53.3%	54.1%	0.8%
8x100M	WPS	1	4	DB	16/32/64/128	1	256	0	6.7%	4.7%	-2.0%
								1	13.3%	12.1%	-1.2%
								2	26.7%	28.0%	1.3%
								3	53.3%	55.2%	1.9%
8x100M	WPS	1	4	DB	16/32/64/128	2	256	0	6.7%	6.8%	0.1%
								1	13.3%	13.8%	0.5%
								2	26.7%	27.8%	1.1%
								3	53.3%	51.6%	-1.7%
8x100M	WPS	1	12	DB	16/32/64/128	2	256	0	6.7%	6.8%	0.1%
								1	13.3%	13.6%	0.3%
								2	26.7%	27.2%	0.5%
								3	53.3%	52.4%	-0.9%
8x100M	WPS	1	4	M	7/7/7/7	3	256	0	25.0%	25.1%	0.1%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WPS	1	4	DB	7/7/7/7	3	256	0	25.0%	24.9%	-0.1%
								1	25.0%	24.8%	-0.2%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.2%	0.2%
8x100M	WPS	1	32	T	7/7/7/7	3	256	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WPS	1	4	DB	25/25/25/25	2	256	0	25.0%	25.8%	0.8%
								1	25.0%	24.7%	-0.3%
								2	25.0%	24.7%	-0.3%
								3	25.0%	24.9%	-0.1%

Chip	Mode	Chan	Streams	Packet Size Code	Weights	L:W Ratio	Frames/ Stream	P	Expected BW %	Actual BW %	Delta BW %
8x100M	WPS	1	32	T	25/25/25/25	2	256	0	25.0%	25.0%	0.0%
								1	25.0%	25.0%	0.0%
								2	25.0%	25.0%	0.0%
								3	25.0%	25.0%	0.0%
8x100M	WPS	1	4	T	15/30/45/60	4	256	0	10.0%	10.0%	0.0%
								1	20.0%	21.0%	1.0%
								2	30.0%	31.9%	1.9%
								3	40.0%	37.1%	-2.9%
8x100M	WPS	1	12	T	15/30/45/60	4	256	0	10.0%	9.8%	-0.2%
								1	20.0%	20.0%	0.0%
								2	30.0%	30.2%	0.2%
								3	40.0%	40.0%	0.0%

References

[1] PMC-Sierra Inc., *PM3370 Ethernet Switch Port Controller Standard Product Data Sheet (PMC-1970861),* PMC-Sierra Inc, 1999.

[2] PMC-Sierra Inc., *PM3380 Ethernet Switch Port Controller Standard Product Data Sheet (PMC-1970861)*, PMC-Sierra Inc, 1999.

[3] IEEE , *IEEE 802.1p - Standard for Local and Metropolitan Area Networks - Supplement to Media Access Control (MAC) Bridges Traffic Class Expediting and Dynamic Multicast Filtering.*

[4] A. Demers, S. Keshav, and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” *Internetworking: Research and Experience*, vol. 1, no. 1, April 1990, pp. 3-26.

[5] PMC-Sierra Inc., *PM3380 Ethernet Switch Port Controller Standard Product Data Sheet (PMC-1970861),* PMC-Sierra Inc, 1999, p. 35.

[6] J. Nagle, “On Packet Switches With Infinite Storage,” *IEEE Transactions on Communications COM-35*, April 1987, pp. 435-438.

[7] B. Beusaou, K.T. Chan, and D.H.K. Tsang, “Credit Based Fair Queuing (CBFQ): A simple and feasible scheduling algorithm for packet networks,” in *Proceedings of the IEEE ATM Workshop*, Portugal, May 1997, pp. 589-594.