QUEUE SCHEDULER VERIFICATION

by

Alison Xu
Bachelor of Applied Science in Computer Engineering
University of Waterloo 2000

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Spring 2005

APPROVAL

Name:	**Alison Xu**
Degree:	**Master of Engineering**
Title of Project:	**Queue Scheduler Verification**

Examining Committee:

Chair: **Dr. Albert Leung**
Graduate Chair, School of Engineering Science, Simon Fraser University

---

Dr. Rick Hobson
Senior Supervisor
Professor, School of Engineering Science, Simon Fraser University

---

Mr. Richard Tse
Industry Supervisor
Technical Advisor, PMC-Sierra Inc.

Date Defended/Approved: April 4, 2005

ABSTRACT

This paper presents the complete verification process of Queue Scheduler. Queue Scheduler is an ASIC function block within PMC-Sierra's PM7354 S/UNI DUPLEX GE, a low cost ATM-to-Ethernet interworking chip for IP DSLAM. It is responsible for fairly scheduling traffic from up to 600 queues, going to up to two destination interfaces. It uses a fair queue scheduling algorithm, programmable for different applications. It is vitally important to verify Queue Scheduler's functional correctness thoroughly, in order to minimize the risk of doing a revision to reduce the overall development cost and to maintain time to market. This paper starts by detailing Queue Scheduler's design specification. After explaining the verification flow, it gives a detailed description of its testbench and testbench components. Next, it discusses the verification test points extracted from the design specification, and the resulting testcases. Finally, it presents the verification results.

DEDICATION

To my God, parents, and husband for their love and support that carried me through the Master of Engineering program.

ACKNOWLEDGEMENTS

I would like to thank the following people for providing me with the knowledge, education, and support that I needed to complete this degree: Dr. Marek Syrzycki for sharing his expertise in the field of analog and digital design; Dr. Kris Iniewski for explaining how telecommunication networks are designed and built; Dr. Ljiljana Trajkovic for teaching how each network protocols work; Dr. Gary Robinson for providing project management techniques; Dr. Glenn Chapman for introducing the field of IC fabrication; Dr. John Dill for showing how data can be better presented and visualized; and the last but not the least, Dr. Rick Hobson and Mr. Richard Tse for their invaluable guidance throughout, without them the completion of my Masters of Engineering Project is not possible.

TABLE OF CONTENTS

Approval ii

Abstract iii

Dedication iv

Acknowledgements v

Table of Contents vi

List of Figures viii

List of Tables ix

Glossary x

1 Introduction 1

2 PM7354 S/UNI DUPLEX GE Overview 2

3 Queue Scheduler TSB Overview 6

4 Queue Scheduler TSB Functional Timing 12

4.1 Queue Status Interface 13
4.2 Queue Request Interface 14
4.3 Port Selection Status Interface 17
4.4 FIFO Status Interface 18
4.5 Port Polling Interface 19

5 Queue Scheduler TSB Operation 21

5.1 Scheduling Mode 21
5.1.1 Queue Contiguous Scheduling Mode 21
5.1.2 Port Contiguous Scheduling Mode 21
5.1.3 Port Interleave Scheduling Mode 22
5.2 Port Priority 22
5.3 Class Selection Calendar 22
5.4 Class Selection Calendar in Conjunction with Class Priority 24
5.5 Class Selection Using Strict Class Priority 25
5.6 Class Promotion 25
5.7 Port Selection Calendar 26
5.8 Bandwidth Limiter 28
5.9 Queue Selection Algorithm 30

5.10 ECBI 31

6 Verification Flow 32

6.1 Planning 32
6.2 Design 32
6.3 Implementation 32
6.4 Verification 33

7 Testbench 34

7.1 ECBI_TBC 34
7.2 Clock Generator 36
7.3 Queue Manager (QMGR) 36
7.3.1 Packet Data Structure 37
7.3.2 Queue Data Structure 37
7.3.3 Port Data Structure 38
7.3.4 Queue Activation Generator 38
7.3.5 Queue Request Responder 41
7.3.6 Bandwidth Monitor 42
7.3.7 Port Poll Response Monitor 42
7.3.8 Queue Selection Checker 43
7.3.9 Statistics 48
7.4 OXA3 49
7.4.1 Destination Interface FIFO 49
7.4.2 Port FIFO 51
7.4.3 Port Poll Response Calendar 51

8 Testcases 53

9 Verification Coverage 57

10 Contribution 61

11 Conclusion 62

References 63

LIST OF FIGURES

Figure 1: Modular GE Uplink DSLAM 4
Figure 2: Fixed Remote DSLAM 5
Figure 3: Ethernet to ATM Data Path 10
Figure 4: Queue Scheduler TSB Block Diagram 11
Figure 5: Queue Status Interface 14
Figure 6: Queue Request Interface – Scenario 1 15
Figure 7: Queue Request Interface - Scenario 2 16
Figure 8: Queue Request Interface - Scenario 3 17
Figure 9: Port Selection Status Interface 18
Figure 10: FIFO Status Interface 19
Figure 11: Port Polling Interface 20
Figure 12: Queue Scheduler TSB Testbench 35
Figure 13: QMGR Queue Selection Algorithm Checker 44

LIST OF TABLES

Table 1: Data Segment Format 9
Table 2: The QS TSB External Interface Signals 12
Table 3: Control Data 14
Table 4: Class Selection Calendar Configuration Example 23
Table 5: Class Selection Calendar Servicing Opportunity Distribution 23
Table 6: Class Selection Calendar Minimum Servicing Opportunity 24
Table 7: Port Selection Calendar Example 1 27
Table 8: Port Selection Calendar Example 2 28
Table 9: Bandwidth Limiter Configuration Examples 30
Table 10: Queue Scheduler TSB Code Coverage Summary 58

GLOSSARY

Acronyms	**Description**
ADSL	Asymmetric Digital Subscriber Line
ASSP	Application Specific Standard Product
ATMIN	ATM In Sub-system
ATMOUT	ATM Out Sub-system
CPE	Customer Premise Equipment
DSLAM	Digital Subscriber Line Access Multiplexer
DUT	Device Under Test
ECBI	Extended Common Bus Interface
ESSIN	Ethernet In Sub-system
ESSOUT	Ethernet Out Sub-system
EOP	End of Packet
GE	Gigabit Ethernet
GES	Gigabit Ethernet Serial
GEP	Gigabit Ethernet Parallel
IFID	Interface Identification
IP	Internet Protocol
OAM	Operation, Administration, and Maintenance
PL2	POS PHY Level 2
QMGR	Queue Manager
QoS	Quality of Service
QS	Queue Scheduler
RTL	Register Transfer Level
SFP	Small Form-Factor Pluggable
SLA	Service Level Agreement
SOP	Start of Packet
SHIM	A component inserted at an interface between two other components

Acronyms	**Description**
TB	Testbench
TBC	Testbench Component
TMSS	Traffic Management Sub-system
TSB	Telecom Standard Block
UL2	Utopia Level 2
VDSL	Very High Speed Digital Subscriber Line

1 INTRODUCTION

The overall cost of developing a semiconductor chip is on the rise due to the ever-increasing design complexity, technology cost, and fabrication cost. Therefore, in order to maintain a chip's time to market, and keep the overall development cost within budget, it is important to achieve first revision success. In order to reach that goal, it is vitally important to verify the chip functional correctness thoroughly, before sending it for fabrication. An RTL fix is much cheaper than a metal fix or a revision. Design verification has become a necessity.

A complex semiconductor chip is usually divided into ASIC blocks by functions for development. Each ASIC function block is firstly verified at the block level. This is called block level verification. After all the ASIC function blocks are designed, implemented and verified, they are stitched together to complete the whole chip. The verification is then performed at the chip level. This is called chip level or top-level verification. Queue Scheduler (QS) is such an ASIC function block, also know as Telecom Standard Block (TSB) within PMC-Sierra. It is a block within PMC-Sierra's PM7354 S/UNI DUPLEX GE, a low cost ATM-to-Ethernet interworking chip for Internet Protocol (IP) Digital Subscriber Line Access Multiplexer (DSLAM). This paper details the QS TSB's complete verification process, in the hope of offering readers a glimpse into the real world of block level design verification.

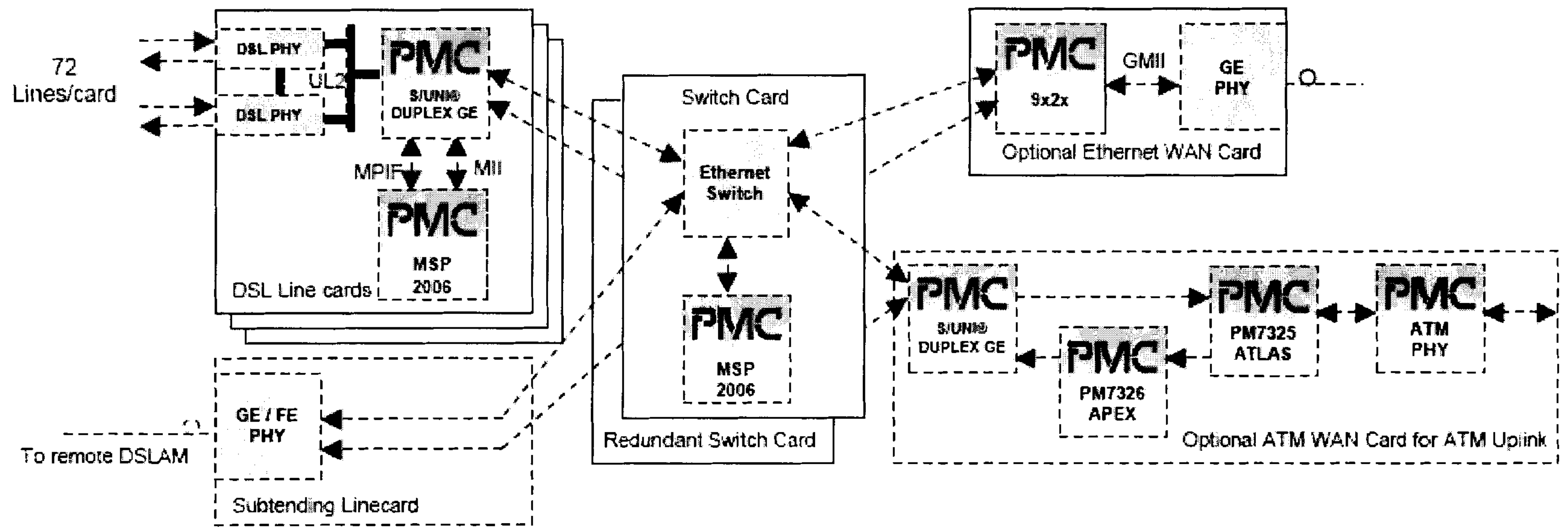
2 PM7354 S/UNI DUPLEX GE OVERVIEW

Following is a brief introduction to PM7354 S/UNI DUPLEX GE to help explaining the QS TSB's purpose and application. PMC-Sierra's PM7354 S/UNI DUPLEX GE is a low cost ATM-to-Ethernet interworking chip for IP DSLAMs. As the ATM networks are migrating to Ethernet-based IP networks, service providers are deploying DSLAMs that can interface to the ATM-based Customer Premise Equipment (CPE) on one side and the Ethernet access network on the other. PM7354 S/UNI DUPLEX GE implements ATM to Ethernet conversion in dedicated hardware, a function deployed to date in expensive network processors or FPGAs. This significantly reduces line card cost that is a major portion of the IP DSLAM bill of materials.

PM7354 S/UNI DUPLEX GE's Multiple Protocol Bus (MPB) interface can be configured to Utopia Level 2 (UL2) interface for connection to ADSL/ADSL2/ADSL2+ modems. Alternatively, MPB interface can be configured to POS PHY Level 2 (PL2) interface for connection to VDSL modems. The MPB interface allows DSLAM vendors to deploy PM7354 S/UNI DUPLEX GE based system with a mix-and-match of ADSL and VDSL line cards. PM7354 S/UNI DUPLEX GE's integrated dual Gigabit Ethernet SERDES interfaces provide redundancy and backplane driving capabilities. PM7354 S/UNI DUPLEX GE can aggregate up to 267 lines of traffic. It offers the following attractive features: full GE line rate throughput, advanced quality of service (QoS), non-blocking multicast, and up to eight priority classes of service simplify end-customer service level agreement (SLA) management and offer further upgrade flexibility for advanced CPEs capable of providing voice, video, and data services. In addition, its user configurable data encapsulation schemes ensure the reusability of the PM7354 S/UNI DUPLEX GE based IP DSLAM line-cards across various geographic locations. Figure 1 shows an application of PM7354 S/UNI DUPLEX GE inside a Modular GE Uplink DSLAM, terminating ATM traffic from a CPE on line cards, and connecting to the off-the-shelf Ethernet switches through its GES interface. Figure 2 shows an application of PM7354 S/UNI DUPLEX GE in Fixed Remote DSLAM, directly connected to Small Form-Factor Pluggable (SFP) optics.

PM7354 S/UNI DUPLEX GE chip is designed in 0.13-micron CMOS technology and packaged in a 352 ball, 27x27mm HSPBGA package. It is a highly complex device with many features and capabilities, explaining it thoroughly is beyond the scope of this paper. In the following sections, only chip information that is crucial to the understanding of the QS TSB function will be presented further.

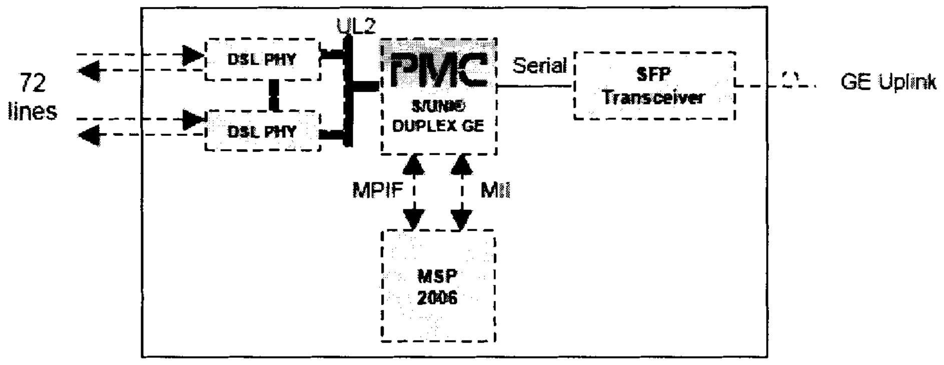
Figure 1: Modular GE Uplink DSLAM



[1]

[1] http://www.pmc-sierra.com/pressRoom/pdf/20050221_fig1.pdf

Figure 2: Fixed Remote DSLAM



2

[2] http://www.pmc-sierra.com/pressRoom/pdf/20050221_fig1.pdf

3 QUEUE SCHEDULER TSB OVERVIEW

QoS involves three sets of parameters: delay, cell loss, and source traffic rate [12]. There are three main types of network traffic: video, voice, and data. Different traffic types have different QoS requirements. Typically video and voice traffic demand low transmission latency, and high bandwidth throughput, but they can tolerate some transmission errors. Data traffic demands small data loss rate, but it is not as demanding as video and voice traffic on transmission delay and bandwidth throughput. Previously there exist separate video, voice, and data network infrastructures. In recent years, service providers are developing a single common network infrastructure where video, voice, and data traffic share an integrated network. The QS TSB contributes to make this convergence a reality. It is responsible for providing software configurable QoS aware egress traffic scheduling. For example, video traffic is classified to be high priority, and data traffic is classified to be low priority (this is not a function of the QS TSB). The QS TSB can be configured to round robin schedule lines of video traffic whenever they become available, and to round robin schedule data traffic in the absence of video traffic. A user can have any type of traffic by using different port and class specifications. The QS TSB's powerful and programmable scheduling algorithm (e.g., bandwidth limiter, user programmable class calendar, and port calendar) can satisfy virtually any QoS requirement of traffic in different applications.

The QS TSB function can probably be best explained by following an Ethernet frame [9] [10] entering PM7354 S/UNI DUPLEX GE at GE Serial 0 (GES0) ingress interface, and exiting as an ATM Utopia Level 2 (UL2) protocol cell [11] at Multiple Protocol (MPB) egress interface (Figure 3). An Ethernet frame is received at GES0 ingress interface. It first goes through the Ethernet In Sub-system (ESSIN), which consists of Enet In block and Ethernet to AAL5 IWF block. ESSIN extracts from the Ethernet frame: destination interface id, port id, and class id. In the case of PM7354 S/UNI DUPLEX GE, there are three destination interfaces: GES, GE parallel (GEP), and MPB egress interface. In this example, the destination interface id of the Ethernet frame is MPB egress interface. ESSIN then performs AAL5 segmentation on the frame and sends the resulting data segment(s) to traffic management sub-system (TMSS). TMSS' QMGR TSB writes the received data segment(s) to external RAM for storing into the ATM bound queue, and sends the queue status information to the QS TSB. Assuming MPB egress interface is configured to UL2, ATM OUT Sub-system (ATMOUT) polls the external PHY layer devices to determine if there is room in them to receive more data. The port polling results are sent back to the QS TSB. The QS TSB makes request for a data segment based on the following parameters:

- Data availability in the QMGR TSB queues
- Space availability in the destination FIFOs within ATMOUT
- Port polling responses received at UL2 or other interfaces. This can be disabled.
- Class selection
- Port selection
- Bandwidth limiter

The requested data segment is later taken from the ATM bound queues and sent directly to the ATMOUT. ATMOUT re-maps the segments to UL2 protocol cell format and sends it out on MPB egress interface.

Due to the versatile nature of PM7354 S/UNI DUPLEX GE, there are many other data paths within. Regardless of the data paths that traffic goes through, the function that QS performs is the same. There are three instances of the QS TSB: QS (GES), QS (GES, GEP), and QS (ATM), altogether they schedule all the queues' traffic in PM7354 S/UNI DUPLEX GE (Figure 3). QS (GES) is responsible for scheduling traffic output bound for GE Serial (GES) egress interface; QS (GES, GEP) is responsible for scheduling traffic output bound for only GEP egress interface, and both GES and GEP egress interfaces; QS (MPB) is responsible for scheduling traffic output bound for MPB egress interface. The QS TSB is designed to satisfy the requirements of all three scenarios.

The QS TSB is responsible for scheduling traffic for up to two destination interfaces. Traffic is stored as data segments in queues. (In the case of PM7354 S/UNI DUPLEX GE, data segments are stored in external SDRAM off the device.) Table 1 shows the data segment format. A segment is always 64 bytes long. Every queue is identified by a port id, a class id, and a destination interface id (IFID). The QS TSB maintains queue status, external interface FIFO threshold, destination port poll status when enabled, and per port bandwidth usage when per port max bandwidth limiter is enabled. Based on the information, along with a user programmable class scheduling algorithm, and a user programmable port scheduling calendar, it fairly schedules among queues of traffic. Once the QS TSB selects a queue, the selected queue sends one data segment. Data segments do not go through the QS TSB. The QS TSB only looks at EGRESS_DATA[4:0] when EGRESS_DSTART is logic 1. EGRESS_DATA[4:0] carries control data bits from word 0 of a data segment: IFID[2:0], EOP, and OAM. The

QS TSB supports three configurations: 75 ports x 8 classes per port, 147 ports x 4 classes per port, and 267 ports x 2 classes per port. Overall, it supports traffic from up to maximum 600 queues. Figure 4 is QS top-level functional block diagram.

Table 1: Data Segment Format

Word	Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0
0	Res | IFID | ClassID	PortID/McGID	Res. | LPS | VALID_DATA_CNT	ERR | EOM | OAM | Res. | DP
1	Reserved	ECC	H1	H2
2	H3	H4	HEC/UDF1	UDF2
3	P1	P2	P3	P4
14	P45	P46	P47	P48
15	CRC 16 (A)		CRC 16 (B)	

Figure 3: Ethernet to ATM Data Path

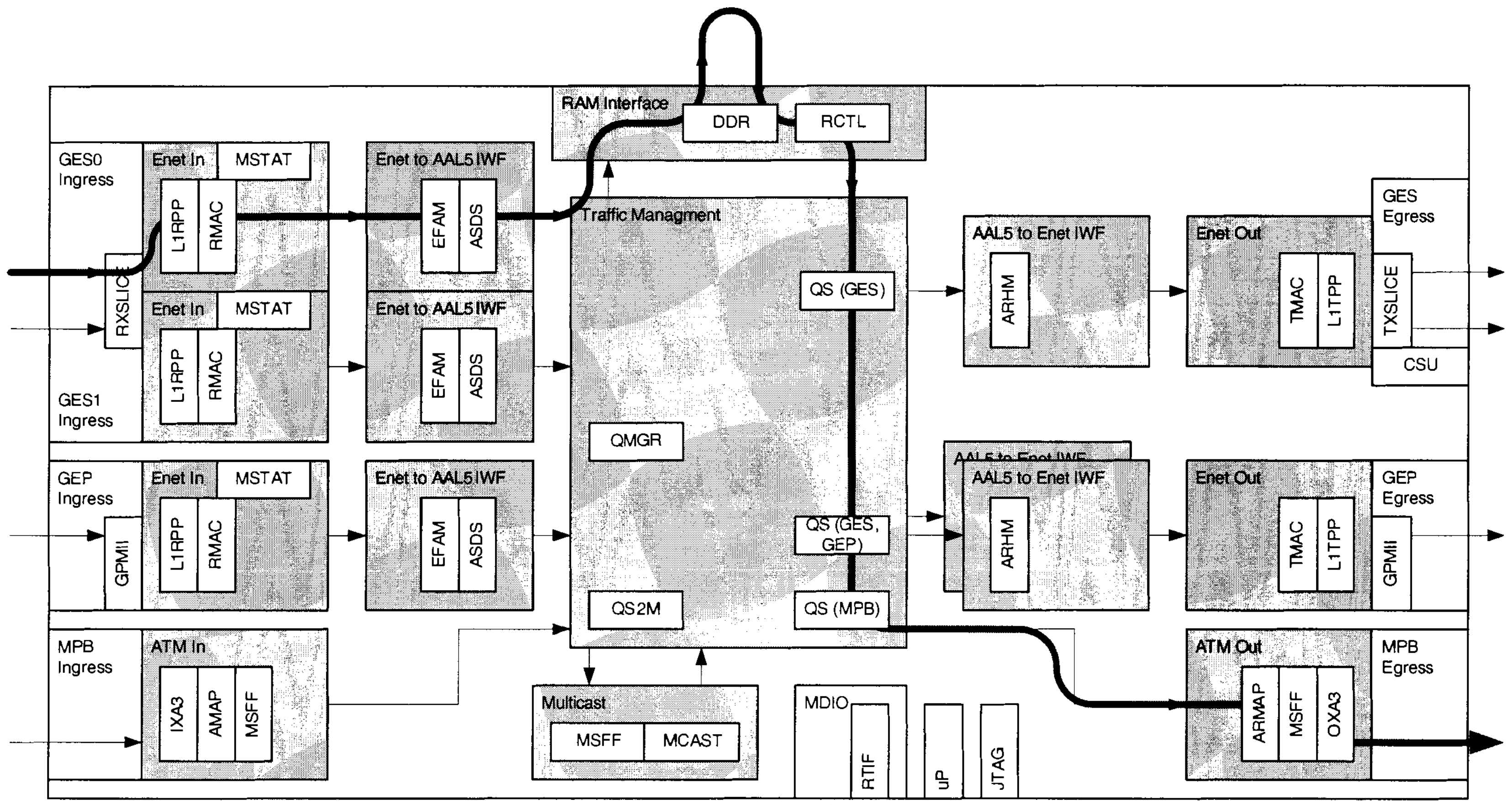
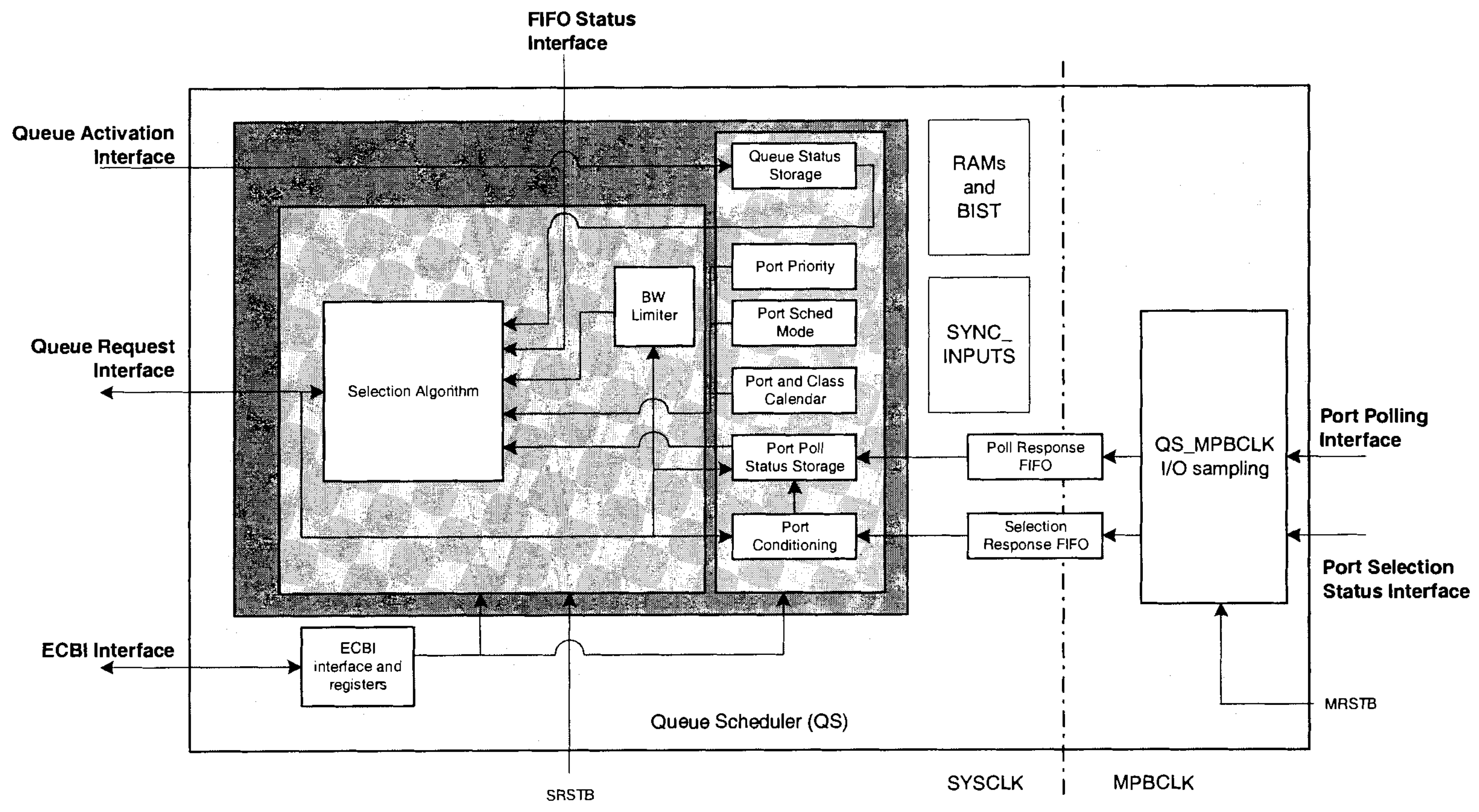


Figure 4: Queue Scheduler TSB Block Diagram



4 QUEUE SCHEDULER TSB FUNCTIONAL TIMING

Table 2 lists the QS TSB's main external interface signals grouped by interface. IFID1[2:0] and IFID2[2:0] specify the two destination interfaces' id. If the QS TSB is responsible for scheduling traffic to only one destination interface, IFID1 and IFID2 should be set to the same value. PORT_CFG[1:0] defines the QS TSB configuration:

- 2'b00 - 75 port x 8 classes per port
- 2'b01 - 147 port x 4 classes per port
- 2'b10 or 2'b11 - 267 port x 2 classes per port

The rest of the signals are described in their respective interface functional timing section.

Table 2: The QS TSB External Interface Signals

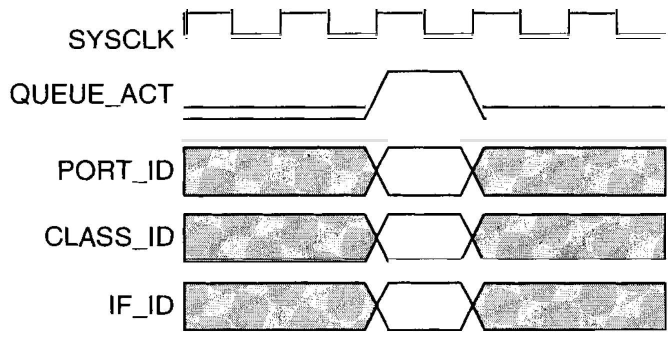
Name	**Type**
General Purpose Interface	
SYSCLK	Input
MPBCLK	Input
PORT_CFG[1:0]	Input
IFID1[2:0]	Input
IFID2[2:0]	Input
Queue Status Interface	
QUEUE_ACT	Input
PORT_ID[8:0]	Input
CLASS_ID[2:0]	Input
IF_ID[2:0]	Input
Queue Request Interface	

Name	**Type**
QS_REQ	Output
QS_REQ_PORT_ID[8:0]	Output
QS_REQ_CLASS_ID[2:0]	Output
QS_ACK	Input
QUEUE_DEACT	Input
QS_ACK_ERR	Input
EGRESS_DSTART	Input
EGRESS_DATA[4:0]	Input
Port Selection Status Interface	
AFULL_1	Input
AFULL_2	Input
FIFO Status Interface	
POLL_RESP_ADDR[8:0]	Input
POLL_RESP	Input
Port Polling Interface	
POLL_RESP_ADDR[8:0]	Input
POLL_RESP	Input

4.1 Queue Status Interface

The QS TSB receives queue activation through Queue Status Interface. Figure 5 is the functional timing diagram of this interface. QUEUE_ACT is set to logic 1 to notify the QS TSB that the queue identified by port id, class id, and interface id as specified on PORT_ID, CLASS_ID and IF_ID has data available. Queue activation can occur at any time.

Figure 5: Queue Status Interface



4.2 Queue Request Interface

The QS TSB schedules traffic through Queue Request Interface. Figure 6, Figure 7, and Figure 8 are the functional timing diagrams of the interface.

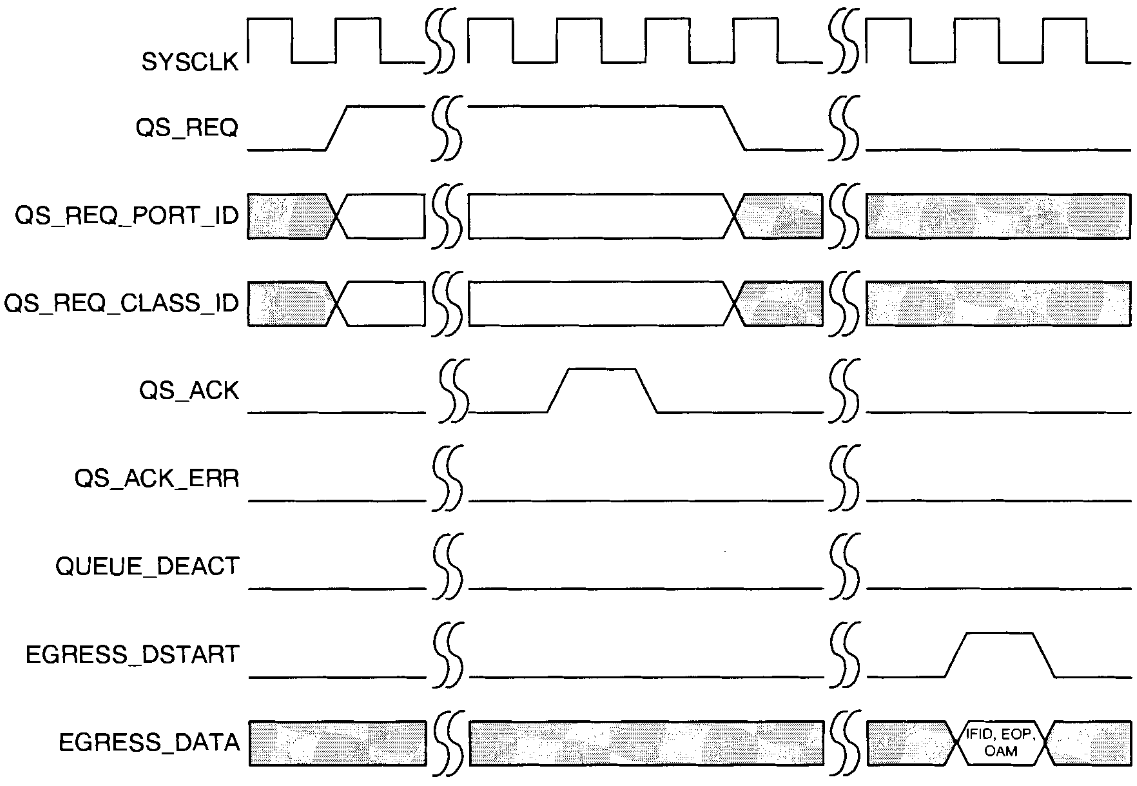
By setting QS_REQ to logic 1, the QS TSB issues a request for a data segment transfer from the queue identified by port id and class id on QS_REQ_PORT_ID and QS_REQ_CLASS_ID. The QS TSB will not issue another request until the current request is acknowledged and the corresponding control data is received. QS_ACK is set to logic 1 to acknowledge the QS TSB queue request. EGRESS_DSTART is set to logic 1 to notify the QS TSB that the control data is valid on EGRESS_DATA[4:0] (Table 3).

Table 3: Control Data

EGRESS_DATA Field	Control Data
EGRESS_DATA[2:0]	IFID[2:0]
EGRESS_DATA[3]	EOP
EGRESS_DATA[4]	OAM

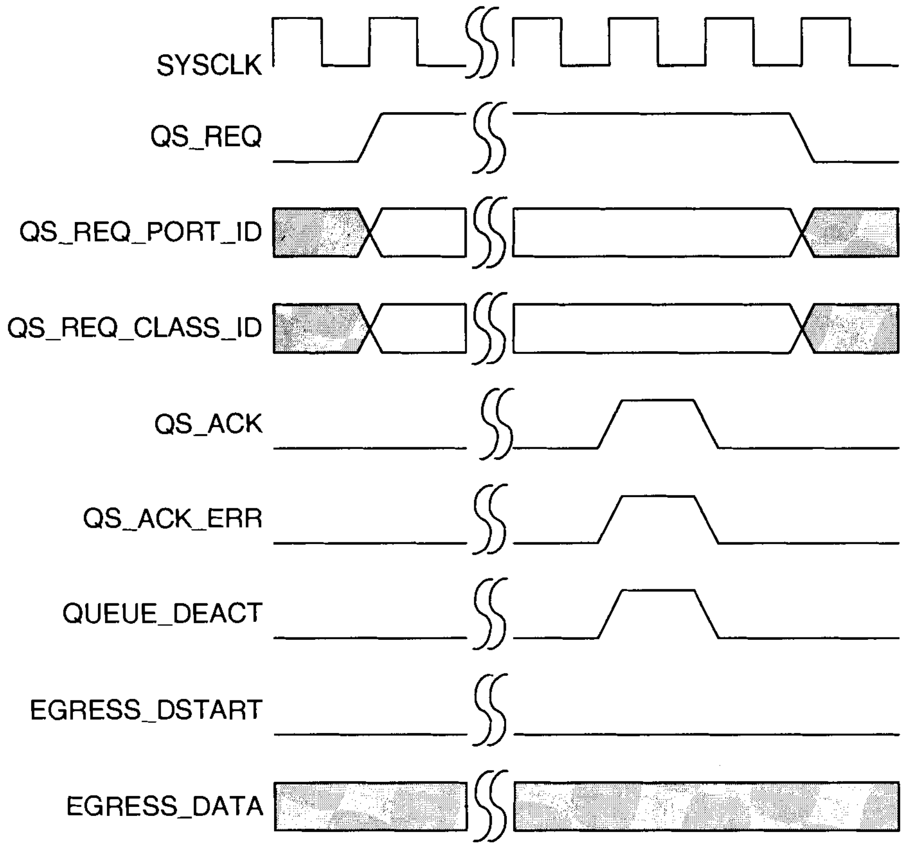
Control data identifies whether the segment is an end of packet (EOP) segment; whether the segment is an operation, administration, and maintenance (OAM) segment; and the destination interface the requested segment is for. The QS TSB only looks at control data with matching IFID, namely traffic that it is responsible for.

Figure 6: Queue Request Interface – Scenario 1



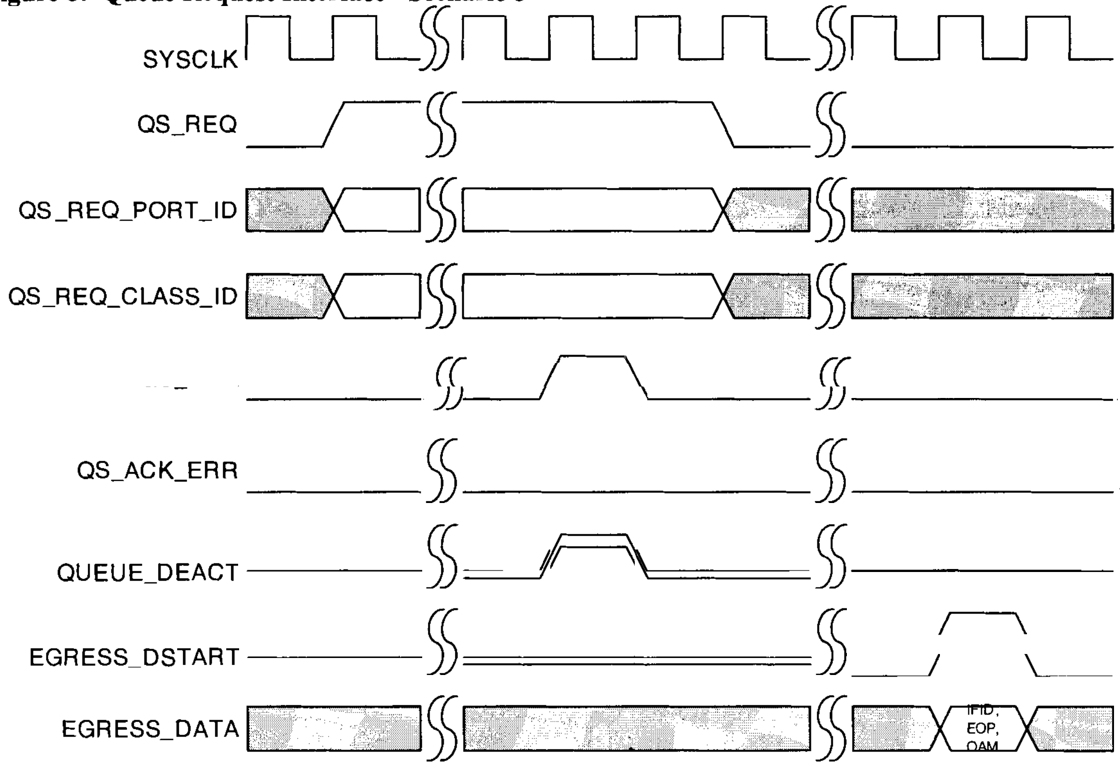
If the QS TSB ever makes a request for a queue that does not have any data, QS_ACK_ERR will be set to logic 1 at the same time as the QS_ACK is set to logic 1. In addition, QUEUE_DEACT will be set to logic 1 indicating the queue identified by port id and class id on QS_REQ_PORT_ID and QS_REQ_CLASS_ID is empty. Under these circumstances, no control data will be transferred. The QS TSB can issue another data segment transfer request immediately.

Figure 7: Queue Request Interface - Scenario 2



If the QS TSB request results in emptying a queue, the request acknowledgement will be accompanied by QUEUE_DEACT being set to logic 1. This indicates that the queue identified by port id and class id on QS_REQ_PORTID and QS_REQ_CLASS_ID has only one remaining data segment and will be empty once this request is serviced. In this case, the QS TSB cannot make another request until it receives the control data for the current request.

Figure 8: Queue Request Interface - Scenario 3

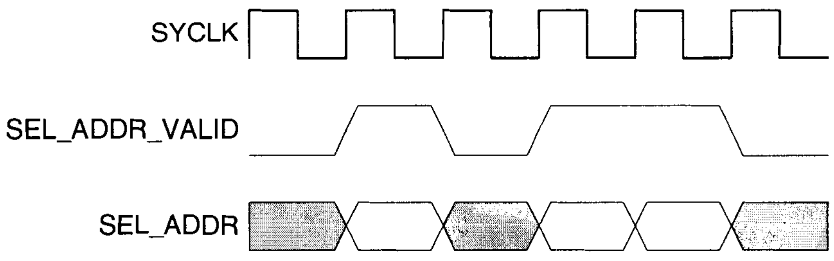


In all cases, both QS_ACK_ERR and QUEUE_DEACT are only valid when QS_ACK is logic 1.

4.3 Port Selection Status Interface

The QS TSB receives the indication of the completion of a segment transfer through Port Selection Status Interface. Figure 9 is the functional timing diagram of this interface. When SEL_ADDR_VALID is logic 1, it indicates the completion of a data segment transfer to the external interface for the port specified on SEL_ADDR. This interface is only used when port polling is enabled. When a segment is requested and acknowledged for a port, that port's polling response will be conditioned (forced to be negative). The completion of the segment transfer, as indicated by this interface will clear the polling conditioning for the port.

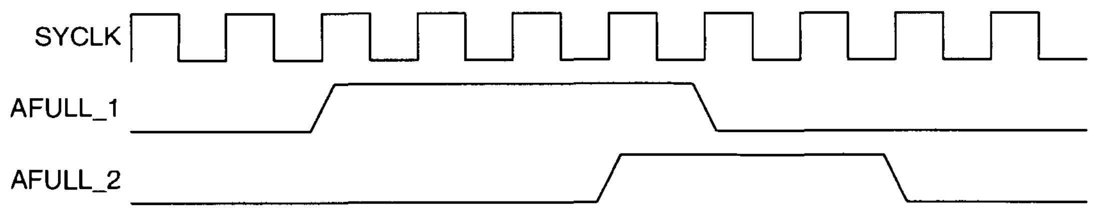
Figure 9: Port Selection Status Interface



4.4 FIFO Status Interface

The QS TSB receives destination interface FIFO fill level through FIFO Status Interface. Figure 10 is the functional timing diagram of this interface. AFULL_1 (almost full) indicates when the destination interface FIFO associated with IFID1 has reached its programmable almost full threshold. Logic 1 indicates that the almost full threshold has been reached or exceeded. Logic 0 indicates the almost full threshold has not been reached. Similarly, AFULL_2 is an indication of the external interface FIFO level associated with IFID2. There are two exceptions. Firstly, when the QS TSB input IFID1 equals IFID2, only AFULL_1 is used and AFULL_2 is ignored. Secondly, if IFID2 is set to 'b111 (both), when scheduling traffic with IFID set to 'b111, since the traffic goes to both destination interfaces, it needs to consider both AFULL_1 and AFULL2. If both IFID1 and IFID2 are set to 'b111, then only AFULL_1 is used and AFULL_2 is ignored. Exception case 1 takes precedence. Each AFULL flag can be completely independent of the other. The QS TSB only makes a request for a data segment from a queue if its destination interface FIFO AFULL flag is not logic 1.

Figure 10: FIFO Status Interface

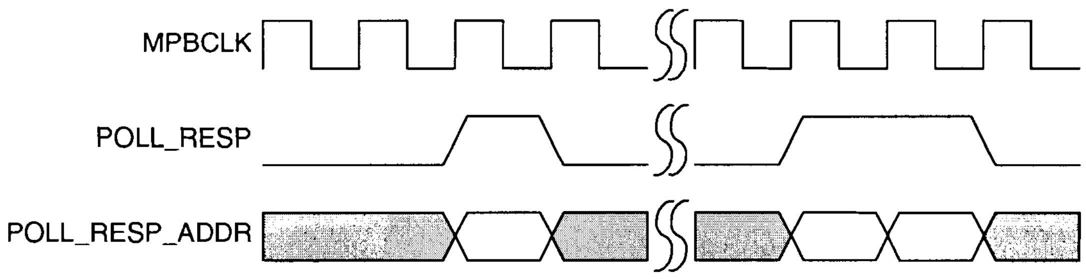


4.5 Port Polling Interface

The QS TSB receives port poll response through Port Polling Interface. Figure 11 is the functional timing diagram of this interface. POLL_RESP is set to logic 1 to notify the QS TSB that the destination port specified on POLL_RESP_ADDR has room for one data segment. There are only positive port polling responses. The QS TSB does not issue port poll request. The destination interface controls the polling and tells the QS TSB which ports have returned positive polling results. This interface is used only when polling is enabled.

When port polling is enabled, the QS TSB provides port poll response condition per port from the time a request for a data segment transfer is made until the time the data segment starts to be transferred on Port Selection Status Interface. During this time port poll response for the given port is ignored. For example, the QS TSB receives a positive port poll response for port 1, i.e. the destination interface FIFO can accept at least one segment worth of data. The QS TSB knows that port 1 class 0 has data, so it issues a data request. The QS TSB ignores all port poll response for port 1 until the requested segment for port 1, class 0 arrives at Port Selection Status interface.

Figure 11: Port Polling Interface



5 QUEUE SCHEDULER TSB OPERATION

5.1 Scheduling Mode

The QS TSB can be configured to operate in three different scheduling modes: queue contiguous, port contiguous, and port interleave.

5.1.1 Queue Contiguous Scheduling Mode

When the QS TSB is configured to operate in queue contiguous scheduling mode, it remains on the selected queue until the end of packet. For example, only port 0 class 0 has data available. Thus, the QS TSB selects port 0 class 0. Later data becomes available for port 0 class 1, port 1 class 0, port 1 class 1. The QS TSB will keep selecting port 0 class 0, i.e. it will not select any other queues, until it sees port 0 class 0's EOP.

5.1.2 Port Contiguous Scheduling Mode

When the QS TSB is not configured to operate in queue contiguous scheduling mode, each port can be individually configured to be either port contiguous or port interleave. When a port is port contiguous, the selected class within the port can only change at the end of packet boundaries, but between data segments, another port maybe selected. For example, only port 0 class 0 has data available. Thus, the QS TSB selects port 0 class 0. Later data becomes available for port 0 class 1, port 1 class 0, port 2 class

1. The QS TSB can select port 1 class 0 or port 2 class 1 at the next selection opportunity. However, it cannot select port 0 class 1, until it sees port 0 class 0's EOP.

5.1.3 Port Interleave Scheduling Mode

When the QS TSB is not configured to operate in queue contiguous scheduling mode, each port can be individually configured to be either port contiguous or port interleave. When a port is port interleave there are no restrictions on when the selected class within the port can change. For example, only port 0 class 0 has data available. Thus, the QS TSB selects port 0 class 0. Later data becomes available for port 0 class 1, port 1 class 0, port 1 class 1. The QS TSB can select any of them at the next selection opportunity.

5.2 Port Priority

A port is configured as either a high priority port or a low priority port via ECBI registers. The QS TSB always schedules a high priority port queue over a low priority port queue.

5.3 Class Selection Calendar

The class selection calendar is enabled when CLASS_CAL_EN is set to logic 1. The class selection calendar allows for servicing opportunity distribution among the classes. The class selection calendar is configured via ECBI registers. The class selection calendar contains 32 class entries. Each class selection calendar entry contains either a valid class valued in the range of:

- 75 port x 8 classes per port mode – class 0 to 7
- 147 port x 4 classes per port mode – class 0 to 3
- 267 port x 2 classes per port mode – class 0 to 1

or a null class value. A null class value is any number outside the valid class value range. The class selection calendar defines the order in which classes will be selected. The class selection calendar can thus guarantee the minimum percentage of servicing opportunity offered to a class. Consider the class selection calendar configuration in Table 4. Please note that there are four null entries. Table 5 shows the resulting class servicing opportunity distribution.

Table 4: Class Selection Calendar Configuration Example

Entry #	Entry Value	Entry #	Entry Value
0	Null	16	null
1	Null	17	4
2	0	18	null
3	2	19	6
4	3	20	5
5	4	21	3
6	5	22	4
7	6	23	6
8	1	24	1
9	4	25	5
10	5	26	2
11	6	27	6
12	2	28	3
13	3	29	4
14	5	30	5
15	6	31	6

Table 5: Class Selection Calendar Servicing Opportunity Distribution

Class	Number of Entries in the Class Selection Calendar	Class Servicing Opportunity
0	1	1/28 = 3.6%
1	2	2/28 = 7.1%
2	3	3/28 = 10.7%
3	4	4/28 = 14.3%
4	5	5/28 = 17.9%
5	6	6/28 = 21.4%

Class	Number of Entries in the Class Selection Calendar	Class Servicing Opportunity
6	7	7/28 = 25%
7	0	0/28 = 0%

5.4 Class Selection Calendar in Conjunction with Class Priority

The QS TSB allows one class of traffic to supersede the class selection calendar. When CLASS_CAL_EN and CLASS_PRI_EN are both set to logic 1, the class defined by PRI_CLASS[2:0] will become a super class and take precedence over the class selection calendar. In the example shown previously, note that there is no entry for class 7 in Table 4. If CLASS_PRI_EN is set to logic 1, and PRI_CLASS[2:0] is set to 7, class 7 traffic, whenever it is available, will be chosen at the next selection opportunity. Namely, super class overrides the class selection calendar. Because of this, super class (class 7 in this example) should not be in the class selection calendar. Moreover, with CLASS_PRI_EN set to logic 1, the minimum servicing opportunity guaranteed to a class via the class selection calendar will depend on the number of servicing opportunities used by the super class. Subtract the super class' maximum bandwidth from the interface's total, the remaining service opportunities can be used to find each class' guaranteed minimum servicing opportunity. Now consider the case where an interface can support 800Mpbs, and class 7 maximum bandwidth is 100Mbps. The remaining minimum bandwidth is 700Mpbs. Continuing the previous example, the resulting guaranteed minimum servicing opportunity for each port is calculated and summarized in Table 6.

Table 6: Class Selection Calendar Minimum Servicing Opportunity

Class	Class Servicing Opportunity	Guaranteed Minimum Servicing Opportunity

Class	Class Servicing Opportunity	Guaranteed Minimum Servicing Opportunity
0	3.6%	3.6% x 700Mpbs = 25.2Mbps
1	7.1%	7.1% x 700Mpbs = 49.7Mbps
2	10.7%	10.7% x 700Mpbs = 74.9Mbps
3	14.3%	14.3% x 700Mpbs = 100.1Mbps
4	17.9%	17.9% x 700Mpbs = 125.3Mbps
5	21.4%	21.4% x 700Mpbs = 149.8Mbps
6	25%	25% x 700Mpbs = 175Mbps
7	0%	unconstrained super class, in the example 100Mbps

5.5 Class Selection Using Strict Class Priority

If CLASS_CAL_EN is set to logic 0, class selection calendar is disabled, class selection is performed using strict class priority algorithm. Traffic from a certain class will not be serviced until all available traffic from higher classes is serviced. When the QS TSB is configured to 75 port x 8 classes per port, class 7 is the highest priority and class 0 is the lowest priority. When the QS TSB is configured to 147 port x 4 classes per port, class 3 is the highest priority and class 0 is the lowest priority. When the QS TSB is configured to 267 port x 2 classes per port, class 1 is high priority and class 0 is low priority.

5.6 Class Promotion

When a port is configured to operate in port contiguous scheduling mode with strict priority class selection, class promotion prevents lower class traffic head-of-line blocking higher class traffic within the same port. Traffic from a lower priority class that is already being serviced is promoted to a priority equal to that of the highest priority class traffic in that port. The head-of-line blocking issue arises because class selection is done before port selection. It is best illustrated using an example. Only port 0 class 0 has data available for transmission. Thus, QS selects port 0 class 0, i.e. port 0 class 0 is in progress. Later data becomes available for port 0 class 1 and port 1 class 1. Because QS is in port contiguous scheduling mode, port 0 class 1 cannot be serviced until port 0 class 0 sees a packet boundary. Because of strict priority class selection, port 0 class 0 cannot be serviced until port 1 class 1 is serviced. Namely, port 0 class 0 is subjecting port 0 class 1 to head-of-line blocking. To improve the latency on servicing the higher class data (port 0 class 1 in this example), class promotion can be enabled. Consequently, port 0 class 0 is promoted to class 1 temporarily. The QS TSB alternatively services between port 1 class 1 and port 0 class 0. As a result, the actual higher class - port 0 class 1 can be serviced in a more timely fashion.

Class promotion can only be enabled when the QS TSB is configured to port contiguous scheduling mode. It should not be enabled when class selection calendar is enabled. It is automatically disabled internally when the QS TSB is configured to queue contiguous scheduling mode

5.7 Port Selection Calendar

The port selection calendar allows servicing opportunity to be distributed among the ports. The port selection calendar is configured via ECBI registers. It contains 512 port entries. Each calendar entry contains either a valid port number in the range of:

- 75 port x 8 classes per port mode – port 0 to 74

- 147 port x 4 classes per port mode – port 0 to 146
- 267 port x 2 classes per port mode – port 0 to 266

or a null port value. A null port value is any number that is outside the valid port range. The contents of the calendar selection entries define the order and frequency in which a port will be selected. Consider the case where round robin selection between 75 ports is desired. The simplest port selection calendar configuration to achieve this would be a calendar with entries 0 through 74 programmed with port number 0 to 74, and the remainder calendar entries programmed to null port value (Table 7).

Table 7: Port Selection Calendar Example 1

Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value
0	0	18	18	36	36	54	54	72	72
1	1	19	19	37	37	55	55	73	73
2	2	20	20	38	38	56	56	74	74
3	3	21	21	39	39	57	57	75	null
4	4	22	22	40	40	58	58	.	.
5	5	23	23	41	41	59	59	.	.
6	6	24	24	42	42	60	60	.	.
7	7	25	25	43	43	61	61	.	.
8	8	26	26	44	44	62	62	.	.
9	9	27	27	45	45	63	63	.	.
10	10	28	28	46	46	64	64	.	.
11	11	29	29	47	47	65	65	.	.
12	12	30	30	48	48	66	66	.	.
13	13	31	31	49	49	67	67	.	.
14	14	32	32	50	50	68	68	.	.
15	15	33	33	51	51	69	69	.	.
16	16	34	34	52	52	70	70	.	.
17	17	35	35	53	53	71	71	511	Null

Now consider the case where an interface can support 800Mbps, 15 ports require 50Mbps while 5 ports require 10Mbps, and each servicing opportunity enables the transfer of a constant sized data segment. The port selection calendar configuration shown in Table 8 will meet the above requirement. H1 to H15 are 50Mbps ports. L1 to

L5 are 10Mbps ports. In this example, only 80 calendar entries are used. Each of the ports requiring 50Mpbs has 4 entries. Each of the ports requires 10Mpbs have 1 entry. Therefore, if the total bandwidth of the interface is 800Mbps, port H1 to H15 will each have 800Mbps * 5/80 = 50Mbps and port L1 to L5 will each have 800Mbps * 1/80 = 10Mbps.

Table 8: Port Selection Calendar Example 2

Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value	Entry #	Entry Value
0	H1	16	H1	32	H1	48	H1	64	H1
1	H2	17	H2	33	H2	49	H2	65	H2
2	H3	18	H3	34	H3	50	H3	66	H3
3	H4	19	H4	35	H4	51	H4	67	H4
4	H5	20	H5	36	H5	52	H5	68	H5
5	H6	21	H6	37	H6	53	H6	69	H6
6	H7	22	H7	38	H7	54	H7	70	H7
7	H8	23	H8	39	H8	55	H8	71	H8
8	H9	24	H9	40	H9	56	H9	72	H9
9	H10	25	H10	41	H10	57	H10	73	H10
10	H11	26	H11	42	H11	58	H11	74	H11
11	H12	27	H12	43	H12	59	H12	75	H12
12	H13	28	H13	44	H13	60	H13	76	H13
13	H14	29	H14	45	H14	61	H14	77	H14
14	H15	30	H15	46	H15	62	H15	78	H15
15	L1	31	L2	47	L3	63	L4	79	L5

5.8 Bandwidth Limiter

There is a bandwidth limiter inside the QS TSB. When it is enabled, it limits any port from exceeding its set maximum bandwidth usage. Once a port reaches its set maximum bandwidth usage, the QS TSB will not schedule traffic for the port until its current bandwidth measuring period has expired.

The bandwidth limiter is usable only for port contiguous and port interleaved mode of operation. It is recommended to disable bandwidth limiter for queue contiguous mode of operation. When the QS TSB is queue contiguous, the outgoing data stream is typically not expected to stop until after an Ethernet frame has completed. However, the bandwidth limiter is not aware of Ethernet frame boundaries. If bandwidth limiting is asserted for a port while it is in the midst of transferring an Ethernet frame, the data transfer will stop. This may cause an under-run in the destination interface FIFO and corrupt the outgoing data stream.

The bandwidth limiter performs bandwidth usage measurement over a specified period of time (measurement period). Whenever a port requests a segment from a queue, its usage level (TX_LVL) is incremented by one. Once the TX_LVL value is equal to the maximum segment transmit value (MAX_LVL), traffic to that port will be stopped. When the specified measurement period ends, the TX_LVL value is reset to zero. This sets a hard-limit on the throughput of the port. The bandwidth limiter has the following parameters:

- BW_LIMIT_EN: User programmed. To enable bandwidth limiter set it to logic 1. To disable bandwidth limiter set to logic 0. This one bit affects all ports in the QS TSB.
- SEGMENT_SIZE: It is a constant value of 64 bytes (Table 1). This value is needed to calculate the actual bandwidth limit in Mbps.
- UNIT_PERIOD: It is a constant value of 2136 SYSCLK cycles, which is determined by the bandwidth limiter implementation. This value is needed to calculate the actual bandwidth limit in Mbps.
- MAX_LEVEL[4:0]: The number of segment transfers allowed in a measurement period before traffic is stopped. Value is in units of segments. It is a constant value of 0x11 (decimal 17).

- TX_LVL[4:0]: Internal count value that represents usage level for a particular port. This will be incremented for a port each time a segment is transferred for that port.
- MP_SETTING[15:0]: The user-programmed measurement period. Value is in number of UNIT_PERIODs. This value is programmable for each port. The configured values should be based on each port's max bandwidth. Minimum value is 0x1. Value of 0x0 is equivalent to 0x1. Maximum value is 0xFFFF.

Table 9 shows bandwidth limiter configuration examples. Assuming SYSCLK is 200 MHz, UNIT_PERIOD = 2136 SYSCLK * 5 ns = 10.68 us.

Table 9: Bandwidth Limiter Configuration Examples

MP_SETTING	Maximum Bandwidth Usage
0x0001	(17 segments * 64 bytes/segment * 8 bits/byte) / (1 * 10.68 us) = 815 Mbps
0x0004	(17 segments * 64 bytes/segment * 8 bits/byte) / (4 * 10.68 us) = 204 Mbps
0x0010	(17 segments * 64 bytes/segment * 8 bits/byte) / (16 * 10.68 us) = 51 Mbps
0xFFFF	(17 segments * 64 bytes/segment * 8 bits/byte) / (65535 * 10.68 us) = 0.0125 Mbps

5.9 Queue Selection Algorithm

The QS TSB queue selection algorithm can be summarized as follows:

- First, high or low port priority is selected using strict priority.
- Next, a class is selected using either the class selection calendar, or strict class priority.
- Finally, a port is selected using the port selection calendar.

5.10 ECBI

All of the configurations described above except PORT_CFG, IFID1, and IFID2 are done through writing to ECBI registers. ECBI registers are also used to monitor the operation of the QS TSB. They are implemented in a configurable and reusable block called Extended Common Bus Interface (ECBI).

6 VERIFICATION FLOW

Verification is a process used to demonstrate functional correctness of a design. The verification of a design usually follows the flow synopsis below.

6.1 Planning

After the preliminary design specification is completed, the first verification phase starts – verification planning. It consists of the following main tasks:

- Verification specification extraction from the design specification
- Preliminary test point list creation
- Verification strategy determination

6.2 Design

The second verification phase is Design. The goal of this phase is to create detailed paper design of testbench (TB) and testbench components (TBC).

6.3 Implementation

The third verification phase is Implementation. The goal of this phase is to implement TB and TBCs designed during the Design Phase. This phase results in coded TB and TBCs ready for verifying the DUT.

6.4 Verification

The final verification phase is Verification. The goal of this phase is to develop, run and debug all testcases to cover all the test points defined in the planning phase. The whole verification suite must be run on the final version of the RTL.

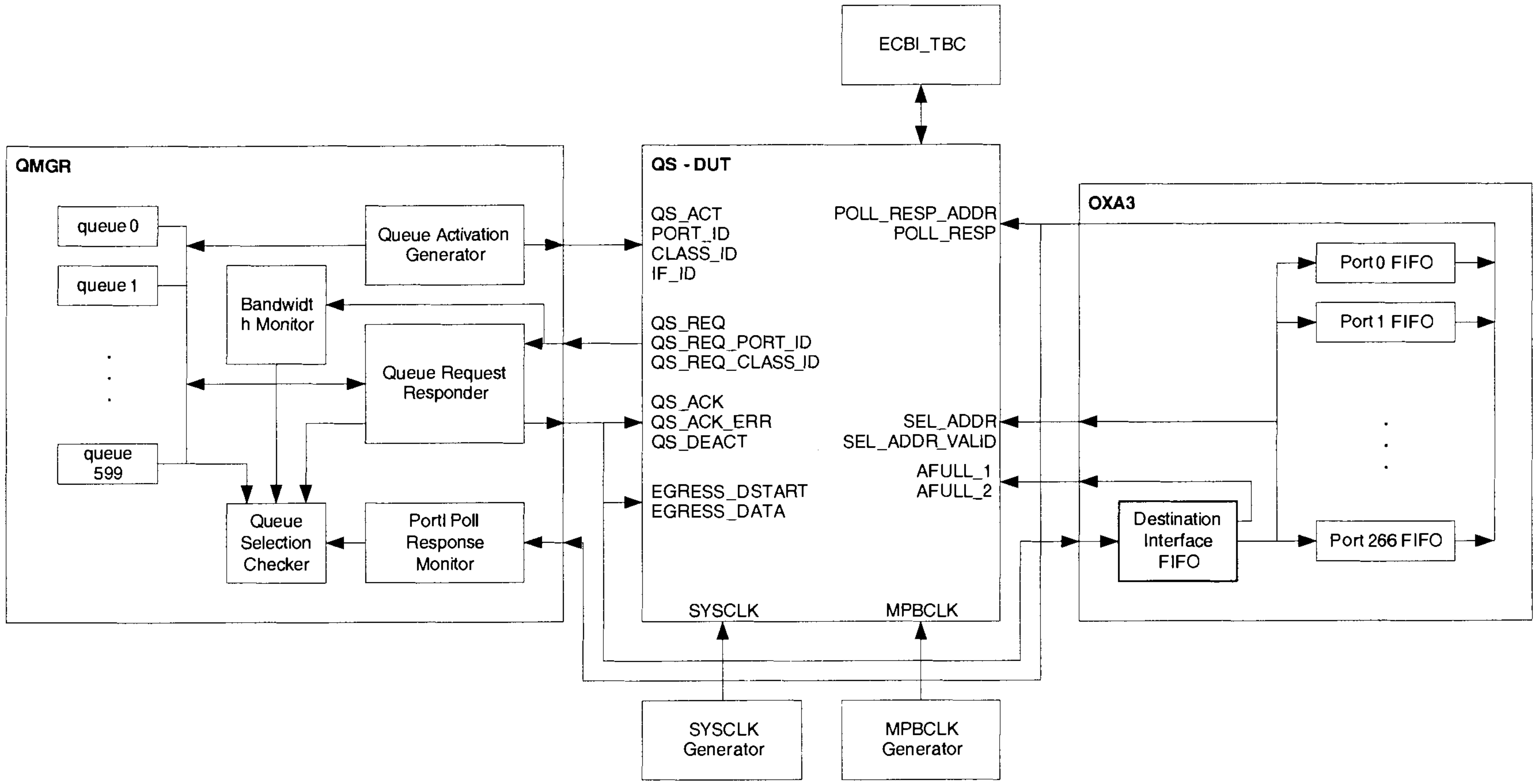
7 TESTBENCH

The QS TSB is verified following the verification flow described in section 6. The verification testbench (Figure 12) consists of the QS TSB, which is the device under test (DUT) and the following TBCs: ECBI_TBC, Clock Generator, Queue Manger, and OXA3. VHDL is the chosen implementation language for the QS TSB. Specman e is the chosen implementation language for TBCs and testcases.

7.1 ECBI_TBC

ECBI has an accompanying user configurable TBC called ECBI_TBC. A user can write and read the QS TSB ECBI registers through it. It is a thoroughly verified TBC. It has been used in many testbenches in verifying various TSBs with ECBI.

Figure 12: Queue Scheduler TSB Testbench



7.2 Clock Generator

Clock Generator is a TBC responsible for driving a clock signal. A user can specify the following clock parameters: period, duty cycle, enable, and phase, using the Specman e method: set(period : int, duty : int, enable : bit, phase : int). There are two input clock signals to the QS TSB: SYSCLK running at 200MHz, and MPBCLK running at 104MHz. Therefore, there are two Clock Generator instances, one for each clock signal. SYSCLK is used to clock all system logic in the QS TSB. MPBCLK is used to clock all MPB logic in the QS TSB. For example, SYSCLK can be configured as follows:

```
// sysclk: frequency = 200 MHz, period = 5,000 ps
sysclk_drv.set(5000, 50, 1, 1000);
```

7.3 Queue Manager (QMGR)

Queue Manager is a TBC that maintains and provides queue status information to the QS TSB, responds to the QS TSB queue request, and checks the QS TSB queue selection. (It is not QMGR TSB.) It interacts with the QS TSB through all its interfaces. QMGR is built on 3 data structures: packet, port, and queue. It provides 6 main functions to the QS TSB:

- generates queue activation
- responds to queue request
- monitors per port bandwidth usage
- maintains port poll response
- check queue selection
- provides overall traffic statistics

7.3.1 Packet Data Structure

Packet is the data structure that represents traffic. It includes the following data fields:

- Packet length in segment (default: 0)
- EOP (default: 0)
  - 0 – packet is missing EOP segment
  - 1 – packet is complete.
- A list of OAM segment positions within a packet (default: null)

By configuring these three data fields, any type of packet can be created. However, a user should never create instances of packet manually in testcases. She should call send_pkt() or send_data_segment() to generate traffic (section 7.3.4). Packets are dynamically created and deleted by QMGR during simulation.

7.3.2 Queue Data Structure

Queue is the building block that represents a physical queue that contains traffic. It includes the following data fields:

- A list of unscheduled packet (default : null)
- Destination interface id (default : 'b000)
- Poll status (default: 0)
- Qualified queue status (default: 0)
- Total number of packet received (default: 0)
- Total number of packet transmitted (default: 0)
- Total number of segments received (default: 0)
- Total number of segment transmitted (default: 0)

QMGR instantiates an array of 600 queues. Any port class queue can be accessed using its queue id as an array index. The queue id is calculated as:

```
queue_id = port_id * max_class_per_port + class_id.
```

For all the examples used throughout this paper, the QS TSB is configured as follows: IFID1 = IFID2 = ‘b000, PORT_CFG = ’b00, unless specified otherwise. For example, port 2 class 3 queue is a QMGR queue identified by queue_id = 2 * 8 + 3 = 19.

7.3.3 Port Data Structure

Port is the basic building block that representing a collection of queues that have the same port id. It includes the following data fields:

- Port priority [high | low] (default: low)
- Eligible (default: 0)
- EOP (default: 1)
- Class (default: 8)
- Port scheduling mode [port contiguous | port interleave] (default: port interleave)
- Port poll conditioning status (default: 0)

QMGR instantiates an array of 267 ports. Any port can be accessed using its port id as an array index. A user is responsible for configuring port priority, and port scheduling mode for each valid port, so it matches the configuration of the QS TSB. QMGR controls data fields: eligible, EOP, class, and port poll conditioning status dynamically during simulation.

7.3.4 Queue Activation Generator

QMGR provides two methods for a user to generate queue activation and to store data segment(s) in QMGR queues.

- `send_pkt(if_id : uint (bits:3), port_id : uint (bits:9), class_id : uint (bits:3), len : uint, oam : bit)`
- `send_data_segment(if_id : uint (bits:3), port_id : uint (bits:9), class_id : uint (bits:3), eop : bit, oam : bit)`

Once send_pkt() is called, after a user constrainable random number of clock cycles delay, QMGR sends the queue activation to the QS TSB through Queue Status Interface compliant with the functional timing specification described in section 0. PORT_ID, CLASS_ID, IF_ID are driven as specified by input parameter if_id, port_id, and class_id. If if_id matches either IFID1 or IFID2, and packet is for a valid queue, the packet is stored in a QMGR queue. If the input parameter oam is 'b1, the second last segment is always an OAM segment; except when the packet length is 1, the first and only segment is an OAM segment.

For example, a user wants to create a 10 segment long packet destined for port 71 and class 7 of destination interface 'b000, she calls send_pkt(0, 71, 7, 10, 0). Since input parameter if_id matches both IFID1 and IFID2, and port 71 and class 7 is a valid queue, QMGR sends the queue activation to the QS TSB for port 71 and class 0. Furthermore, it stores this 10-segment long non-OAM packet in queue 71 * 8 + 7 = 575. A user can use send_pkt() to send an invalid queue activation to the QS TSB. For example, a user wants to verify that the QS TSB does not schedule traffic for a destination interface that it is not responsible for, she calls send_pkt(5, 0, 0, 10, 0). QMGR sends the queue activation to the QS TSB. Since input parameter if_id matches neither IFID1 nor IFID2, QMGR does not store the packet in any of its queues. If the QS TSB incorrectly makes a queue request responding to the invalid queue activation, QMGR will detect it and assert an error, as the invalid segment the QS TSB requested is not stored in QMGR.

Similarly, once send_data_segment() is called, after a user constrainable random number of clock cycles delay, QMGR sends queue activation to the QS TSB through Queue Status Interface compliant with the functional timing specification. PORT_ID, CLASS_ID, IF_ID are driven as specified by input parameter if_id, port_id, and class_id. If if_id matches either IFID1 or IFID2, and the segment is for a valid queue, it is stored in a QMGR queue. If the queue is empty or the last packet in the queue is complete, a new packet is created and appended to the packet list to store the segment; else the segment is appended to the last incomplete packet (missing EOP). If input parameter eop is ‘b1, the packet’s data field EOP is set. If input parameter oam is 1, the packet’s data field list of OAM position is updated. send_data_segment() gives a user total control in data creation. For example, a user wants to create 3 packets: the first packet is a complete 4 segment long packet with the second and the third segment being OAM segments; the second packet is an 1 OAM segment long packet, and the third packet is a 2 non-OAM segment long partial packet. They are all for destination interface ‘b000, port 0 and class 0. She calls send_data_segment() as follows, assuming the queue is empty initially:

```
// First complete packet
Send_data_segment(0, 0, 0, 0, 0);    // SOP
Send_data_segment(0, 0, 0, 0, 1);    // OAM
Send_data_segment(0, 0, 0, 0, 1);    // OAM
Send_data_segment(0, 0, 0, 1, 0);    // EOP
// Second complete packet
Send_data_segment(0, 0, 0, 1, 1);    // SOP & EOP & OAM
// Third incomplete packet
Send_data_segment(0, 0, 0, 0, 0);    // SOP
Send_data_segment(0, 0, 0, 0, 0);    // Just a segment
```

Traffic from a queue should go to the same destination interface. A user must not call send_pkt() and send_data_segment() with same input parameters class_id and port_id, but different if_id. For example, the QS TSB is configured as follows: IFID1 = ‘b000, IFID2 = ‘b111 (both), and PORT_CFG = ’b00. A user calls send_pkt(0, 7, 7, 6, 0). This stores a six segments long, non-OAM packet in port 7 class 7 queue. This packet is destined to go to IFID1. A user must not call send_pkt() or send_data_segment() with input parameters: if_id = 'b111, port_id = 7, and class_id = 7, as that also stores data in the same queue but destined for both IFID1 and IFID2. This is not a valid case.

7.3.5 Queue Request Responder

QMGR Queue Request Responder monitors the QS TSB Queue Request Interface. It is automatically enabled always. Upon receiving a queue request, it checks the queue requested for the following:

- Is the queue empty?
- Is there only one segment left in the queue?
- Is the segment an EOP segment?
- Is the segment an OAM segment?
- What is the destination interface the traffic within the queue requested destined for?

Base on the information, QMGR drives the QS TSB inputs: QS_ACK, QS_ACK_ERR, and QUEUE_DEACT. After a user constrainable random number of clock cycles delay, it drives the QS TSB inputs: EGRESS_DSTART and EGRESS_DATA. It responds to the QS TSB queue request through Queue Request Interface and Port Selection Status Interface in compliance with the protocol and timing specification described in section 0 and 4.3. There is one exception. When QMGR is enabled to inject QUEUE_DEACT error, it will not assert QUEUE_DEACT when it should. If the QS TSB successfully requests a segment, QMGR removes the segment from its queue. Furthermore, to assist the checking of the QS TSB' queue scheduling algorithm, it records the queue id, and whether the segment is an EOP segment. It clears the poll status of the queue requested. It configures the data fields of the port requested as follows :

- class = class requested
- EOP = whether the segment is an EOP segment
- port poll conditioning status = 1

The checking of the QS TSB queue selection algorithm is explained in section 7.3.8. If the QS TSB makes a request to an empty queue, QMGR asserts an error. For example:

```
[24641500] qmgr_u-@3: ERROR: request data from empty port =
0 class = 0
```

7.3.6 Bandwidth Monitor

The bandwidth monitor measures the QS TSB per port bandwidth usage when enabled. A user should only enable bandwidth monitor, if the QS TSB bandwidth limiter is also enabled. Bandwidth monitor configuration must match that of the QS TSB's bandwidth limiter. Bandwidth monitor asserts a warning when a port exceeds its set maximum bandwidth usage. It is based on the same design as the QS TSB's bandwidth limiter. Bandwidth monitor has the following programmable fields: bw_monitor_en, mp_setting[port]. For example, a user wants to enable bandwidth monitor, and set port 0 measuring period to 10 UNIT_PERIODS, she needs to do the following:

```
qmgr.bw_monitor.bw_monitor_en = 1'b1;
qmgr.bw_monitor.mp_setting[0] = 10;
```

7.3.7 Port Poll Response Monitor

QMGR monitors the QS TSB Port Polling Interface. Whenever a port's poll response is received, it checks to see if the port's poll conditioning status is set. If it is not set, then it sets poll status for all the valid queues within the port. QMGR also monitor the Port Selection Status Interface. Whenever a segment transfer is completed for a port as indicated on Port Selection Status Interface, QMGR clears that port's poll conditioning status. (Whenever the QS TSB successfully requests a segment for a port, the poll statuses of all the queues within that port are cleared, and the port's poll conditioning status is set. This is done by the Queue Request Responder as described previously in section 7.3.5.)

7.3.8 Queue Selection Checker

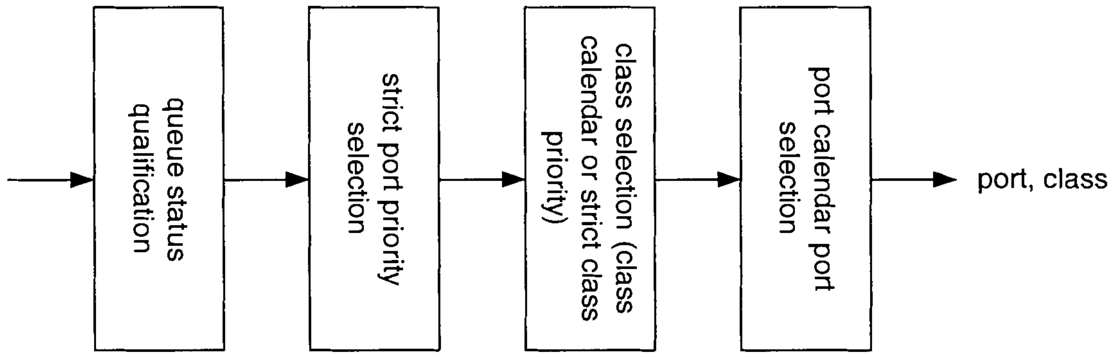
QMGR checks the QS TSB queue selection by monitoring every queue request. Firstly, it checks that the QS TSB does not request traffic to destination interface whose FIFO almost full threshold has been reached or exceeded, as described in section 0. Specifically, if AFULL_1 is logic 1, and the QS TSB issues a request for a queue with traffic going to destination interface IFID1 or IFID2 ('b111 - both), QMGR asserts an error. QMGR also asserts an error, when AFULL_2 is logic 1, and IFID2 does not equal IFID1, and the QS TSB makes a request for a queue with traffic going to destination interface IFID2.

Secondly, QMGR checks that the QS TSB remains on the selected queue until the end of packet, when it is configured to queue contiguous scheduling mode. Specifically, if the last segment requested is not an EOP segment, and the last queue requested is not empty, but the QS TSB requests a segment from a different queue, QMGR asserts an error. QMGR performs this check automatically always.

7.3.8.1 Queue Selection Algorithm Checker

When the QS TSB queue selection algorithm checking is enabled, upon the receipt of a queue request, with the information that it gathers by monitoring various interfaces, QMGR goes through the queue selection algorithm (the same as the one that the QS TSB goes through) to pick a queue. If the queue it picks does not match the queue that the QS TSB is requesting, it asserts a warning. It is important to investigate each queue selection mismatch manually to determine the cause. A mismatch is not always an error, because the checker is not a cycle accurate model of the queue selection engine implemented inside the QS TSB. Figure 13 graphically illustrates the stages that QMGR goes through to pick a queue.

Figure 13: QMGR Queue Selection Algorithm Checker



queue status qualification
strict port priority selection
class selection (class calendar or strict class priority)
port calendar port selection
port, class

7.3.8.1.1 Queue status qualification

Stage 1: QMGR goes through all the queues to determine their queue status. If a queue has segment(s), QMGR sets its queue status; otherwise, it clears its queue status.

Stage 2: Every queue status is further qualified by its destination interface FIFO almost full level. Specifically, if a queue's destination interface is 'b111 (both), and IFID2 is set to 'b111, and IFID2 does not equal IFID1, its status is qualified by both

AFULL_1 and AFULL2; else if a queue's destination interface is IFID1, its queue status is qualified by AFULL_1; else if a queue's destination interface is IFID2, its queue status is qualified by AFULL_2.

Stage 3: If port polling is enabled, every queue's queue status is further qualified by its port's poll status. QMGR sets qualified queue status only if it is set in the last stage, and its port poll status is set.

Stage 4: If neither class promotion is enabled, nor class selection calendar is enabled, for a port that is port contiguous, QMGR clears the qualified queue status of any class higher than the class in progress within that port. For example, port 1's data fields: class = 1, EOP = 0. It means that for port 1, class 1 is the last class that transmitted a data segment, and the segment is not an EOP segment. In other words, port 1 class 0 is in the middle of a packet transmission. As a result QMGR clears all the higher priority classes (2-7) qualified queue statue. Therefore, even if those higher priority classes have data available, they are not eligible to be requested. There is no need to clear the qualified queue status of lower priority classes, since QMGR will pick the class in progress based on class priority.

If class selection calendar is enabled, for a port that is port contiguous, and it has a queue class in progress, QMGR clears the qualified queue status of all the other classes. QMGR does this to stay port contiguous - the selected class within the port can only change at end of packet boundaries.

Stage 5: If bandwidth monitor is enabled, and a port has reached or exceeded its bandwidth usage, QMGR clears the qualified queue status of all the queues within the port.

Stage 6: QMGR goes through all the valid queues, if a port has a queue with qualified queue status set, it sets that port's data field: eligible. It indicates that the port is eligible to be scheduled. In summary, an eligible port satisfies the following conditions:

- The port is not empty.
- Destination interface FIFO is below its almost full threshold.
- Destination port is not busy.
- Port contiguous if configured so.
- Class promotion disabled if configured so.
- Per port bandwidth usage limit is not exceeded.

7.3.8.1.2 Port Priority Selection

QMGR goes through all the high priority ports, if any one of them is eligible (i.e. data field: eligible is set), QMGR selects high priority ports, else it goes through all the low priority ports, if any one of them is eligible, QMGR selects low priority ports. If there is no eligible port, QMGR asserts an error, as this is a mismatch between QMGR and the QS TSB queue selection. Please note in this stage, QMGR only selects the port priority. It has selected neither a class nor a port. Class selection is next.

7.3.8.1.3 Class Selection: class priority

If class selection calendar is not enabled, class selection is based on strict class priority. QMGR goes through all the queue of the selected port priority, and selects the highest priority class that has its qualified queue status set.

7.3.8.1.4 Class Selection: class selection calendar

If both class selection calendar and class priority are enabled, and any priority class queue of the selected port priority has its qualified queue set, QMGR selects the priority class. However if none of the priority class queues of the selected port priority has its qualified queue set, QMGR goes through the class selection calendar starting from where it left off last time it traversed it for the selected port priority. It selects the first class with a queue of the selected port priority that has its qualified queue status set. If after traversing the whole class selection calendar, QMGR cannot find a qualified class, it asserts an error, as this is a mismatch between QMGR and the QS TSB queue selection.

Please note in this stage, QMGR only selects a class. It has not selected a port. Port selection is next.

7.3.8.1.5 Port Selection

QMGR goes through the port calendar starting from where it left off last time it traversed it for the selected port priority. It selects the first port with a queue of the selected port priority and selected class that has its qualified queue status set. If class promotion is enabled, and the queue selected is in a port that has another class queue in progress, QMGR selected the class in progress instead. If after traversing the whole port calendar, QMGR cannot find a qualified port, it asserts an error, as this is a mismatch between QMGR and the QS TSB queue selection.

Finally, QMGR has selected a port and a class. If the queue selected by QMGR does not match the queue requested by the QS TSB, QMGR asserts a mismatch warning.

7.3.8.2 Queue Selection Algorithm Checker Weakness

Given the high complexity in the QS TSB design specification, and stringent timing requirement, the QS TSB is implemented with pipelines, FIFO, RAMs, and other complex logic. As a result, even though the QS TSB and QMGR monitor the same interfaces, there are discrepancies between the time that input information reaches the QS TSB decision logic and QMGR decision logic. For example after the QS TSB receives a port poll response, it writes the information into a FIFO. Later the information is read out of the FIFO and sent to the decision making logic. This timing delay is not fixed. It is a result of many factors. However, QMGR uses the port poll response immediately after receiving it. Therefore, it is possible that QMGR selects a port that the QS TSB still thinks is busy. Considering the available verification resources, the complexity of the task, and the potential benefit, it is not deemed worthwhile to create a cycle accurate queue selection model. However, the weakness of the queue selection checker can be avoided by creating the stimulus carefully, or by making sure that both QMGR and the QS TSB have the same information before being allowed to make a queue selection.

7.3.9 Statistics

QMGR has a method output_statistics(). Once called, it displays the following information for each QMGR queue

- Unscheduled data
- Total number of segment received
- Total number of segment transmitted
- Total number of packet received
- Total number of packet transmitted

This method provides a sanity check for the QS TSB. It is essential to call this method at the end of very testcase. It verifies that the QS TSB is not tuck in an orphan state and has stopped requesting data.

7.4 OXA3

OXA3 is the TBC that drives Port Polling Interface, FIFO Status Interface, and Port Selection Status Interface (Figure 12). It provides a user with the following four methods to control AFULL_1 and AFULL_2:

- Set_afull_1()
- Set_afull_2()
- Clear_afull_1()
- Clear_afull_2()

OXA3 can also automatically control AFULL_1 and AFULL_2 if enabled, based on the fill level of a simple destination interface FIFO. OXA3 allows a user to set port poll status for each port. For example:

```
// set port 0 poll status to logic 0
oxa3.port_status[0] = 1'b0;
// set port 1 poll status to logic 1
Oxa3.port_statu[1] = 1'b1;
```

It can also automatically set port poll status for all the ports, based on the fill level of a simple FIFO inside each destination port.

7.4.1 Destination Interface FIFO

OXA3 observes the Queue Request Interface: EGRESS_DSTART, EGRESS_DATA, and EGRESS_ADDR. Once it receives a valid segment's control information, it creates a virtual segment, and writes it into the destination interface FIFO. It increments destination interface 1 FIFO segment counter. If the segment IFID is both ('b111), OXA3 also increments destination interface 2 FIFO segment counter. The virtual segment contains port id, and a delay of random number of clock cycles. Every MPBCLK cycle, OXA3 decrements the delay of every segment inside the FIFO until it reaches zero. OXA3 reads out the segment at the head of the FIFO , once its delay has expired. It decrements the corresponding segment counter(s). It sets SEL_ADDR_VALID to logic 1, and puts the port id on SEL_ADDR for 1 MPBCLK cycle on Port Selection Status Interface. If error injection on SEL_ADDR is enabled, the LSB of SEL_ADDR is flipped. The destination interface FIFO mimics the segment finally reaching its destination port after going through data path logic and FIFO. Essentially, a segment's delay is the minimum number of MPBCLK cycle that it remains in the FIFO.

If OXA3 automatic AFULL control is enabled, and destination interface 1 segment counter exceeds its FIFO threshold, OXA3 asserts AFULL_1. OXA3 will not de-assert AFULL_1 until the segment counter is below the threshold. Similarly, if destination 2 segment counter exceeds its FIFO threshold, OXA3 asserts AFULL_2. It will not de-assert AFULL_2 until the segment counter is below the threshold. A user can specify the two FIFOs' threshold, and enable automatic AFULL control as follows:

```
// destination FIFO 1 threshold is 4 [segment]
oxa3.fifo1_th = 4;
// destination FIFO 2 threshold is 2 [segment]
oxa3.fifo2_th = 2;
// Enable OXA3 automatic AFULL control
```

```
oxa3.afull_auto = 1'b1;
```

7.4.2 Port FIFO

OXA3 observes the Port Selection Status Interface. Once it receives a valid segment, it writes the segment into the corresponding port's FIFO. A port FIFO drains its segments at a specified drain rate. All port FIFOs have the same drain rate. For example a user can specify the FIFO drain rate as follows:

```
// drain 1 segment every 12 MPBCLK cycles
oxa3.fifo_drain_rate = 12;
```

When a user enables OXA3 to automatically set port poll status, if the number of segments within a port FIFO exceed its threshold, its port poll status is cleared. Its port poll status will not be set until the number of segments within it drops below the threshold. All port FIFOs have the same threshold. For example, a user can specify port FIFO threshold as follows:

```
// port FIFO threshold is 10 [segment]
oxa3.fifo_th = 10;
// Enable OXA3 automatic port poll response
oxa3.poll_status_auto = 1'b1;
```

7.4.3 Port Poll Response Calendar

By default, OXA3 cycles through all the valid ports (from port 0 to port max), outputting their port poll status. A user can customize the port poll response calendar, specifying the length of the calendar and value of each entry. For example, a user wants to create a short calendar outputting the port poll status of only 2 ports: port 0 and port 74:

```
oxa3.max_cal_idx = 1;         // 2 valid port entry
oxa3.poll_resp_cal[0]  = 0;   // first entry is port 0
oxa3.poll_resp_cal[1]  = 74;  // second entry is port 74
```

8 TESTCASES

Test points are extracted from the QS TSB design specification. As the design progresses, the design specification changes. Some features are modified, some features are added, and some features are removed. Test points are updated accordingly. Each test point is covered in at least one testcase. Here is the list of the main test points:

1. Verify all the QS TSB ECBI register bits have a default value that matches the default value specified in the design specification after the QS TSB reset. All readable and writable bits in the QS TSB should be writeable and should hold the value written until the QS TSB is reset or until another value is written. There is no ECBI register address aliasing.
2. Verify port selection using port calendar in 75 port x 8 classes per port configuration
3. Same as above except in 147 port x 4 classes per port configuration
4. Same as above except in 267 port x 2 classes per port configuration
5. Verify class selection using strict class priority in port contiguous scheduling mode in 75 port x 8 classes per port configuration
6. Same as above except in 147 port x 4 classes per port configuration
7. Same as above except in 267 port x 2 classes per port configuration
8. Verify port priority (high/low) selection
9. Verify class promotion when class promotion is enabled in port contiguous mode
10. Verify no class promotion when class promotion is disabled in port contiguous mode
11. Verify bandwidth limiter
12. Verify port priority selection, class selection using strict class priority, and port selection using port calendar together in port contiguous scheduling mode
13. Same as above except in port interleave scheduling mode
14. Same as above except in queue contiguous scheduling mode
15. Verify the QS TSB is able to respond to illegal PORT_ID and CLASS_ID of Queue Activation Interface, POLL_RESP_ADDR of Port Polling Interface, and

SEL_ADDR of Port Selection Status Interface in 75 port x 8 classes per port configuration

16. Same as above except in 147 port x 4 classes per port configuration
17. Same as above except in 267 port x 2 classes per port configuration
18. Verify the QS TSB is able to respond to all legal PORT_ID and CLASS_ID of Queue Activation Interface, POLL_RESP_ADDR of Port Polling Interface, and SEL_ADDR of Port Selection Status Interface in 75 port x 8 classes per port configuration
19. Same as above except in 147 port x 4 classes per port configuration
20. Same as above except in 267 port x 2 classes per port configuration
21. Verify class selection using class selection calendar in port contiguous scheduling mode
22. Same as above except in port interleave scheduling mode
23. Same as above except in queue contiguous scheduling mode
24. Verify class selection using class selection calendar and priority class in port contiguous mode
25. Same as above except in port interleave scheduling mode
26. Same as above except in queue contiguous scheduling mode
27. Verify QS_ACK_ERR assertion (along with QUEUE_ACK) in response to a QS_REQ, with QS_ACK_ERR response latency from 1 clock cycle up to 6 clock cycles in port contiguous scheduling mode.
28. Same as above except in port interleave scheduling mode
29. Same as above except in queue contiguous scheduling mode
30. Verify that the QS TSB is able to differentiate a port configured to port contiguous scheduling mode and a port configured to port interleave scheduling mode when queue scheduling
31. Verify queue contiguous scheduling mode. The QS TSB remains on the selected queue until the end of packet, even if there are segments in a higher priority class.
32. Verify that when IFID1 and IFID2 are different and IFID2 is BOTH, the QS TSB responds to all combinations of AFULL_1 and AFULL2. Send data with IFID equals IFID1 or IFID2
33. Verify the QS TSB qualifies queue status with AFULL_1 and AFULL_2 correctly, with all combinations of IFID1 and IFID2

There are two types of testcases: directed and random. Directed testcases have most of the input stimulus (queue activation, port poll response, FIFO status) well controlled, as a result the simulation outcome is known. The advantages of directed testcases are:

- They are usually not very long.
- They are easy to debug.
- A user can be sure that a test point is covered.

Directed testcases are especially helpful in debugging the QS TSB at its earlier development stage. Furthermore, directed testcases can be carefully setup so that QMGR queue selection algorithm checker when enabled, does not give false alarms about the QS TSB making queue selection incorrectly. All the test points are covered in at least one direct testcase. The drawback of direct testcases is that it covers limited combinations of input stimulus, and it does not stress test the QS TSB over a long period of time.

Random testcases complement directed testcases. Random testcases have input stimulus randomly generated with constraints. They result in a greater number of input stimulus combinations, thus provide more complete verification coverage. The drawback is that QMGR queue selection algorithm checker may raise false alarms (section 7.3.8.2). However, this issue can be overcome by sending queue activations to the QS TSB before it is enabled to schedule traffic, and port poll responses and FIFO statuses do not change dynamically during simulation. For example, in testcase 75_pc_pi_random, the QS TSB is configured as follows:

- IFID1 = IFID2
- 75 ports x 8 classes per port
- odd ports are low priority, even ports are high priority
- port 0 ~ 36 are configured to port contiguous scheduling mode, port 37 ~ 74 are configured to port interleave scheduling mode
- class promotion is enabled

QMGR is programmed to match. OXA3 is programmed to send positive poll response for all valid ports. At the beginning of simulation, AFULL_1 and AFULL_2 are set high. QMGR generates 1000 random queue activations with the following constraints:

- randomly select a port from port 0 ~ 74,
- randomly select a class with the following probability
  class 0 – 25%
  class 1 – 15%
  class 2 – 15%
  class 3 – 15%
  class 4 – 15%
  class 5 – 10 %
  class 6 – 5%
  class 7 – 5%
- randomly select a packet length with the following probability
  1 segment – 50%
  4 to 7 segment – 50%
- OAM – 50%, non-OAM – 50%

After all the queue activations are sent, AFULL_1 and AFULL_2 are set low. The QS TSB starts to issue queue requests. In this testcase QMGR queue selection algorithm checker does the checking reliably when enabled.

If a testcase is completely random: queue statuses, port poll responses, and FIFO statuses are all changing dynamically throughout the simulation, QMGR Queue Selection Checker should not be enabled. QMGR does do the following queue selection check automatically:

- The QS TSB does not issue a queue request for an invalid port id or an invalid class id
- The QS TSB does not issue a request for queue with traffic to a destination interface FIFO that has reached or exceeded it almost full threshold
- The QS TSB is queue contiguous when configured so
- The QS TSB does not issue a queue request for a busy port

Altogether, there are over 40 directed and random testcases.

9 VERIFICATION COVERAGE

After all testcases have been written and run without errors, a code coverage tool is used to get a qualitative measure of the quality of a testbench by collecting statistics of several code coverage criteria. The code coverage tool used is VNavigator from TransEDA. It runs on top of the VHDL – Specman simulator. The following code coverage metrics are collected:

- State coverage – measures the number of executed statements in the code
- Branch coverage – measures the number of branches executed in "IF ELSIF ELSE END IF" and "CASE" blocks. It makes sure both the "TRUE" and "FALSE" branches have been executed.
- Basic Sub-condition Coverage (BSC) – makes sure all sub-condition in a logic statement has been toggled TRUE and FALSE.

The minimum requirement is 95% for all three code coverage metrics.

In general, the higher the code coverage result, the more confident the engineer is about the verification. However, a code coverage tool cannot tell the engineer that she has verified the design functionally 100%. 100% code coverage does not mean the design is 100% bug free. It can help her to identify parts of the circuit that are not exercised by the existing testcases.

Code coverage results are summarized in Table 10. Statement and branch coverage results meet the requirement. However, the condition coverage result does not meet the requirement. Below is the detailed study of the coverage results in order to determine the cause of low condition coverage.

Table 10: Queue Scheduler TSB Code Coverage Summary

Metric		**Coverage**
Statement	1080/1116	96.7%
Branch	603/629	95.8%
Condition	715/758	94.3%

Indirect Interface is a reusable block. It has 28 of 32 sub-conditions covered. The VNavigator found that core_handshake = ‘0’ is never false in the two instances of the following condition statement:

```
   IF ((core_busy_signal = '1') OR (core_handshake = '0'))
 then
```

The RTL was studied. It turns out that core_handshake is an input to the module. It is tied to logic 0 by the calling module. This accounts for the 2 sub-conditions not covered.

Interrupt Clear is a reusable block. It has 8 out of 10 sub-conditions covered. VNavigator found that scanb is never logic 0 in the following 2 statements.

```
din   <= clrint OR scanb;
rstin <= (NOT ((NOT int) AND scanb) ) AND rstb;
```

scanb is a test signal, when set to logic 0, it puts the QS TSB in test mode. In order to put the QS TSB in normal mode for functional verification, it must be set to logic 1. This accounts for 2 sub-conditions not covered.

2 to 4 decoder is a reusable module. It has 6 out 10 sub-conditions covered. VNavigator found that enable is never logic 0 in the following statements:

```
outb(0) <= NOT( selb(1) AND selb(0) AND enable);
outb(1) <= NOT( selb(1) AND sel(0)  AND enable);
outb(2) <= NOT( sel(1)  AND selb(0) AND enable);
```

```
    outb( 3) <= NOT( sel(1)  AND sel(0)  AND enable);
```

Enable is an input signal to the module. The module was instantiated in 2 different places. In both places, enable is tied to logic '1'. This accounts for 4 sub-conditions not covered.

ECBI address decoder is a reusable module. It has 0 out of 24 sub-conditions not covered. VNavigator found that none of the sub-condition is ever true or false in the following statements.

```
    IF cbi_addr_width = 1 GENERATE
     regenb(0) <= addr_sel( 0) OR bsb;
     regenb(1) <= NOT addr_sel( 0) OR bsb;
    END GENERATE;

    IF cbi_addr_width = 4 GENERATE
     decode_enable(0) <= addr_sel( 3) OR bsb;
     decode_enable(1) <= NOT addr_sel( 3) OR bsb;
    END GENERATE;

    IF cbi_addr_width = 7 GENERATE
     decode_enable(0) <= addr_sel( 6) OR bsb;
     decode_enable(1) <= NOT addr_sel( 6) OR bsb;
    END GENERATE;

    IF cbi_addr_width = 10 GENERATE
     decode_enable(0) <= addr_sel(9);
     decode_enable(1) <= not addr_sel(9);
    END GENERATE;
```

This is because it is instantiated with cbi_addr_width set to six. As a result, none of the statements above is actually generated. This accounts for all 24 sub-condition not covered.

In summary there are total (2 + 4 + 24 + 2) = 32 not covered sub-conditions that should not be taken into account when calculating condition coverage, because they are tied to either logic 1 or logic 0 (i.e. can not toggle) for simulation or implementation reasons. Therefore the recalculated condition coverage is:

$$ConditionCoverage = \frac{715}{758 - (2 + 4 + 24 + 2)} = \frac{715}{726} = 98.5\%$$

The QS TSB verification achieves the code coverage goal.

10 CONTRIBUTION

I was not involved in the specification, design, and implementation of the QS TSB. I was responsible for the verification of the QS TSB. It took me 6-months from verification planning, testbench and testbench components design and implementation, to create and execute testcases to verify the QS TSB functional correctness.

11 CONCLUSION

All the verification test points are covered in testcases. All testcases pass on the QS TSB RTL tapeout netlist. All code coverage requirements are met. Therefore, the QS TSB verification is complete. There are no QS TSB bugs found in top-level verification. At the time this paper is written, PM7354 S/UNI DUPLEX GE is in a foundry being fabricated. Whether the QS TSB works in silicon is yet to be determined after the fabricated and packaged chip is tested in the validation lab.

Block level verification requires the block to be verified thoroughly. It covers all the block features and all the corner cases. The requirement and approach of top-level verification is different. Possible future work might be demonstrating the top-level verification process through the example of a chip's top-level verification effort.

REFERENCES

1. PMC-Sierra, Inc. PMC-2040668, "QS Telecom System Block Engineering Document". Issue 2, 2004.

2. PMC-Sierra, Inc. PMC-2040896, "Queue Scheduler Telecom System Block (QS TSB) Register Descriptions". Issue 2, 2004.

3. PMC-Sierra, Inc. PMC ICDC, cad_dd_00377, "Specman Best Practices Document". Issue 2, 2003.

4. PMC-Sierra, Inc. PMC ICDC, cad_dd_00647, "VN_2003.12 Application Note". Issue 1, 2004.

5. Verifica LLC. "Introduction to Specman Elite Verification Methods". Rev 2004-04-09, Portland, Oregon, 2004

6. PMC-Sierra, Inc. "PM7354 S/UNI DUPLEX GE Product Detail", retrieved on March 19, 2005, http://www.pmc-sierra.com/products/details/pm7354/

7. PMC-Sierra, Inc. PMC-2031884, "DUPLEX GE ASSP Device Design Document". Issue 3, 2005.

8. PMC-Sierra, Inc. "PMC-Sierra Unveils Industry-Leading Device for IP DSLAMs To Accelerate Triple Play Services; Low Cost S/UNI DUPLEX GE Based IP DSLAMs Enable Advanced Customer Premises Equipment", retrieved on March 19, 2005, http://phx.corporate-ir.net/phoenix.zhtml?c=74533&p=irol-newsArticle&ID=676972

9. IEEE. 802.3-2000 IEEE Part 3:Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 2000.

10. IEEE. 802.1Q IEEE standards for local and metropolitan area networks. Virtual bridged local area networks, 2003.

11. ATM Forum. UTOPIA Level 2. Version 1.0.af-phy-0039.000, June 1995.

12. Jean Walrand and Pravin Varaiya. High-performance Communication Networks: 2nd Edition. Morgan Kaufmann Publishers, 2000.