WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

# DYNAMIC QUEUE MANAGEMENT

# FOR NETWORK QUALITY-OF-SERVICE

by

YUHUA CHEN

Prepared under the direction of Professor Jonathan S. Turner

A thesis presented to the Sever Institute of

Washington University in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

May, 1998

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

# ABSTRACT

# DYNAMIC QUEUE MANAGEMENT
# FOR ATM NETWORK QUALITY-OF-SERVICE

by YUHUA CHEN

ADVISOR: Professor Jonathan S. Turner

May, 1998
Saint Louis, Missouri

Today, ATM networks are being used to carry bursty data traffic with large and highly variable transmission rates, and burst sizes ranging from kilobytes to megabytes. Obtaining good statistical multiplexing performance for this kind of traffic requires much larger buffers than are needed for more predictable applications or for bursty data applications with more limited burst transmission rates. Large buffers lead to large queueing delays, making it necessary for switches to implement more sophisticated queueing mechanisms in order to deliver acceptable Quality of Service (QoS). This thesis describes a 2.4 Gb/s ATM queue management chip that has practically unlimited buffer scaling and also supports dynamic per VC queueing, an efficiently implementable form of weighted round-robin scheduling, a novel packet-level discarding algorithm and the ability to support multiple output links. We show that with current technology, it is feasible to provide per VC queueing for over 8,000 simultaneously active virtual circuits and over one million cells.

Short Title: Dynamic Queue Management Chen M.S. 1998

# ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Jonathan S. Turner for all his help and guidance in the research work and in the writing of this thesis. I would like to thank the other members of my thesis committee, Dr. Roger Chamberlain, Dr. William Richard, Dr. Fred Rosenberger, and Tom Chaney for their useful comments. I would also like to thank Andy Fingerhut and Margaret Fluke for the helpful discussions.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 1. INTRODUCTION

## 1.1. Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) networks are designed for B-ISDN (Broadband Integrated Service Digital Networks). With ATM, applications with many different data rates, can be sent “asynchronously” using statistical multiplexing, where the system bandwidth is shared by active sources. Data for video-on-demand, live television, CD-quality music and high-speed data transfers, can all be transmitted over the same network.

ATM networks are “connection-oriented”, meaning that a virtual connection needs to be established between communicating terminals before transmission can take place. Connections are distinguished by *Virtual Path Identifiers* (VPI) and *Virtual Circuit Identifiers* (VCI). ATM networks use fixed-length cells containing 5 bytes of header and 48 bytes of payload. There are two types of connections: *Virtual Circuit* (VC) connection and *Virtual Path* (VP) connection. In a virtual circuit connection, cell is switched based on both the VPI and the VCI. In order to add a new connection, routing tables in all switches in the path need to be modified. In a virtual path connection, virtual circuits following the same path are assigned the same VPI. Intermediate switches route cells using the VPI only. Therefore, new VCs can be established over VP connections without modifying routing tables in intermediate switches, simplifying the handling of those virtual circuits. Figure 1-1 shows an ATM network with several virtual circuit connections established among end hosts. The circles represent ATM switches. Connections with the same path are treated as a single virtual path connection illustrated by shaded area. The VPI and VCI of a connection usually change hop by hop so that each switch can pick up an unused (VPI, VCI) pair locally to accommodate a new connection.

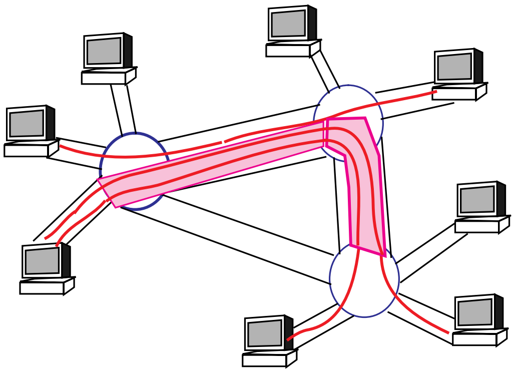


Figure 1-1 ATM Network

The ATM cell formats in both the User-Network Interface (UNI) and the Network-Network Interface (NNI) are shown in Figure 1-2. The only difference between these two formats is that the UNI cell format contains a Generic Flow Control field (GFC). The VPI field is 8 bits and 12 bits in UNI and NNI formats, respectively. A network can

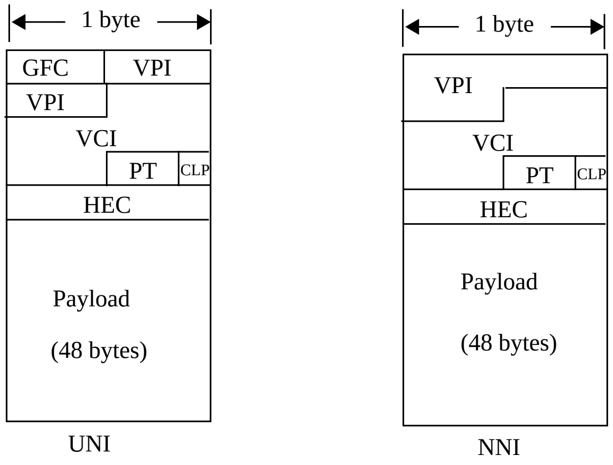


Figure 1-2 ATM Cell Formats

support 256 virtual path connections using UNI format. The VCI field is 16 bits in both formats. Since a virtual circuit connection is determined by both the VPI and the VCI, the possible number of combinations is quite large: $2^{28}$ for the NNI case. Switches generally provide routing table entries for only a small fraction of this number.

The Payload Type (PT) identifies cells as user data or control information. The Cell Loss Priority (CLP) bit is set for low priority cells. The Header Error Check (HEC) implements the error-detecting code on the header. The Generic Flow Control (GFC) field in the UNI format is currently unused. The payload field carries the data and is 48 bytes long.

Because transport layer protocols deal with packets instead of ATM cells, an ATM adaptation Layer (AAL) is defined to allow users to send packets larger than an ATM cell. Packets are segmented into ATM cells, transmitted through the network, and reassembled at the other end. AAL5 uses the U-bit of the PT field to indicate the last cell of a packet.

# 1.2. Switching Systems

## 1.2.1. General Concept

ATM switches are used to route cells arriving on their input ports to the desired output ports. As shown in Figure 1-3, a typical ATM switch consists of three major components: Input Port Processors, Switching Network and Output Port Processors.

The Input Port Processor contains a *Virtual Path/Circuit Translation Table* (VXT), specifying the destination port of connections and the VPI and VCI used by the

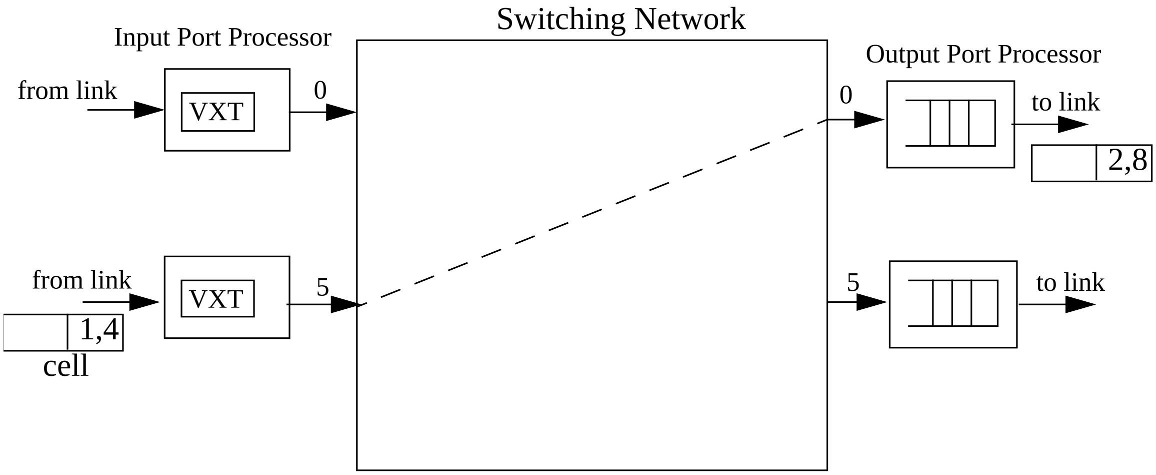


Figure 1-3  ATM Switching System

downstream switch. If a cell belongs to a virtual path connection, the VPI alone is used to select a table entry containing the required outgoing link number and a new VPI. The VCI is unchanged in this case. If the cell belongs to a virtual circuit connection, the cell’s VPI and VCI are used to select a table entry containing the required output link number and new VPI and VCI values.

The Switching Network routes ATM cells to one or possibly several specified output ports. The switching network is the central part of a switching system. The design issues include scalability, reliability and cost-effectiveness.

The Output Port Processor receives cells from the switching network and buffers cells waiting for transmission. Some switch architectures allow the cells to arrive at the OPP out of order. These switch designs require the Output Port Processor to resequence the cells before sending them to the external link.

## 1.2.2. Washington University Gigabit Switch (WUGS)

High speed ATM switching systems are desirable because high speed cores usually have less fragmention of bandwidth, higher utilization of memories, and less system cost per port. The Washington University Gigabit Switch (WUGS) [3] has been designed and implemented to support port speeds up to 2.4 Gb/s. It supports one-to-many, many-to-many, many-to-one multicast in a particularly cost-effective way. The system can be configured from 8 ports to 4096 ports and throughput approaching 10 Tb/s.

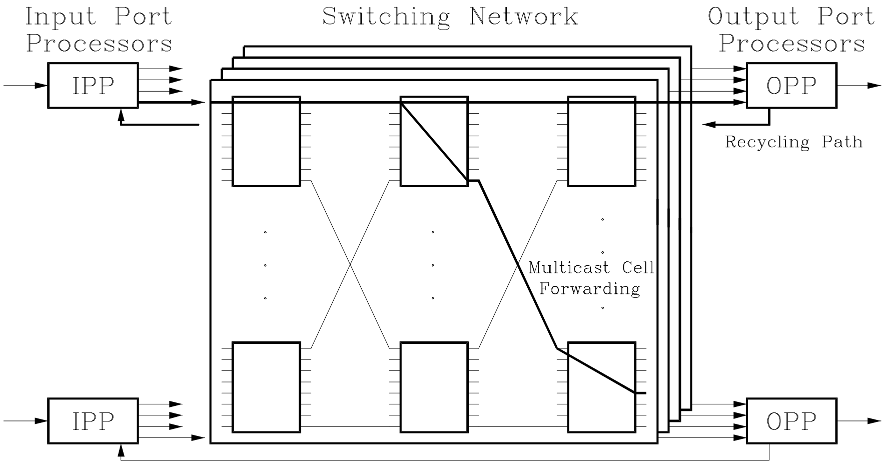


Figure 1-4 WU Gigabit Switch (from [3])

Figure 1-4 shows the overall organization of the WUGS switching system. It consists of three main components, each of which is implemented as a single chip. The *Input Port Processors* (IPP) receive cells from the incoming links, buffer them while awaiting transmission through the central switching network and perform the virtual path/circuit translation required to route cells to their proper outputs. The *Output Port Processors* (OPP) resequence cells received from the switching network and transmit them to the external link. Each OPP is connected to its corresponding IPP, providing the ability to recycle cells belonging to multicast connections. The central switching network is made up of *Switching Elements* (SE) with eight inputs and outputs and a common buffer to resolve local contention. The SEs switch cells to the proper outputs or dynamically distribute cells to provide load balancing.

To provide sufficient bandwidth for 2.4 Gb/s link rates on the external links, the switch carries ATM cells in a 36 bit wide format with a clock rate of 120 MHz. The switching network is implemented in four parallel planes, with each plane receiving the same address information, plus eight bits of data. The cells proceed through the four planes in parallel, without any explicit coordination, and are reconstructed at the OPP.

The OPP and IPP link interfaces use a subset of the functionality defined in the *Universal Test & Operations PHY Interface for ATM* (UTOPIA) 2 standard [1]. The OPP and IPP link interfaces operate in either 16-bit or 32-bit mode as defined by an option pin on each IC. The 16-bit mode and 32-bit mode support transmission at rates up to 1.2 Gb/s and 2.4 Gb/s, respectively.

## 1.3. Motivation

When ATM network technology was first developed in the 1980s, its developers envisioned a comprehensive traffic management methodology, with explicit reservation of resources, end-to-end pacing of user data streams to conform to resource reservations and network-level enforcement mechanisms to protect against inadvertent or intentional violation of resource reservations. In the context of such methodology, efficient statistical multiplexing performance could be achieved without large amounts of buffering in the network and with very simple queueing mechanisms.

As ATM was deployed in the 1990s, the original expectations for traffic management were found to be unrealistic. ATM is now being used largely to support internet data traffic which is highly unpredictable and for which the traffic management philosophy of ATM is difficult to apply. In the current application context, resources are generally not explicitly reserved, end systems do not pace their transmissions and most network equipment cannot enforce resource usage limits. In this environment, to obtain good statistical multiplexing performance and high link utilization, one needs large buffers. In particular, one needs buffers that are at least comparable, and preferably an order of magnitude larger than user data bursts, which range in size from kilobytes to megabytes. Unfortunately, the use of large buffers with simple FIFO queueing disciplines leads to poor performance for real-time traffic and allows “greedy” applications to appropriate an unfair portion of network resources. Providing good *Quality of Service* (QoS) to real-time applications and fair treatment to bursty data applications requires more sophisticated queueing and cell scheduling mechanisms.

### 1.3.1. Accessing High Speed Switches

Gigabit switches are advantageous in many ways. They can naturally handle high data rate applications such as multimedia applications. In addition, they can handle lower rate applications in a very efficient way. The cost of switches are shared by all links, which reduces the cost per port. High bandwidth switching systems reduce delay in the interconnection networks. They provide better queueing performance for bursty traffic and have less fragmentation of bandwidth and memory.

However, since lower speed links are commonly used for applications with lower data rates, it is important to provide mechanisms to allow access to high speed switches from lower speed links. One way to do so is to introduce a cell multiplexor before the Input Port Process (IPP) and a demultiplexor following the Output Port Processor (OPP). The design of the multiplexor is relatively easy if the total link bandwidth does not exceed the bandwidth of the switch. This thesis focuses on the design of the Dynamic Queue Management chip that functions as the demultiplexor between the Output Port Processor and the lower speed links.

### 1.3.2. Tolerating Data Surges

The demultiplexor can be implemented in a simple way if the outgoing links are rarely subject to traffic overload. However, since bursty traffic is common to data applications, the system must have sufficient buffering to accommodate temporary data surges. This is particularly important for lower speed links such as OC-3, where a single application may consume a large fraction of the link's capacity. The Dynamic Queue Management chip provides an efficient way to manage a large buffer space, so that the buffer size is limited only by the memory cost.

### 1.3.3. Providing Quality-of-Service

In the presence of large buffers, Quality-of-Service (QoS) becomes an important issue in the design. Real-time applications, such as video and audio require small end-to-end delay. Cells that arrive late are discarded and no recovery is possible. Real-time traffic is usually continuous stream and has limited variability. Admission control can be used to avoid congestion. Non real-time applications, such as data transfer, are usually bursty. It is less predictable so that congestion can occur routinely. Because of the high peak-to-average ratio, large buffers are needed to avoid cell loss.

Large buffers and priority treatment imply per VC queueing instead of FIFO queueing. In systems with FIFO queueing, cells from different VCs enter the same queue. Large buffers introduce large queueing delays. Most of the delays are caused by "mis-behaving" VCs that send at a higher rate than they should. In system implementing per

VC queueing, cells from different VCs enter separate queues. With proper scheduling mechanisms, queueing delays due to large buffers and “mis-behaving” VCs are eliminated.

In this thesis, we design a *Dynamic Queue Management* (DQM) chip that connects to the output side of a high performance ATM switch, such as the Washington University Gigabit Switch. It supports separate queues for each application data stream and buffer sizes that are limited only by the cost of memory. The design can be implemented with a single application-specific integrated circuit in 0.35 micron CMOS technology together with SRAM components. The design will support a total output rate of 2.4 Gb/s and can support either a single OC-48 link, or a combination of lower speed links.

## 1.4. Related Work

Several queueing and scheduling schemes have been proposed to support Quality-of-Services (QOS) in ATM switch designs.

Four scheduling schemes have been compared by Chipalkatti [8]: FIFO, minimum laxity threshold (MLT), head-of-the-line priority (HOLP) and queue length threshold (QLT). Cells are served in their arrival order in FIFO scheduling and no QoS is supported. For MLT, each queue has a timer. When the timer expires, the corresponding queue sends a service request. QLT scheduling balances the bandwidth utilization by setting thresholds for each queue. In HOLP scheduling, queues with higher priority are served before those with lower priority.

The asynchronous time sharing (ATS) mechanism proposed by Lazar [13] partitions the network into four separate classes. Each class is serviced some minimum portion of a larger frame time so that the bandwidth is guaranteed. Unused guaranteed bandwidth of a given class can be used by the next traffic class.

Hashemi has explored a cell sequencer that inserts new cells into the physical queue according to its class/group and the state of the logical queues already present [10]. Each cell is assigned a tag value according to its priority class before entering the sequencer. The sequencer arranges the cells based on their tag values by sequentially comparing the tag of the new cell with those already in the physical queue.

Landsberg and Zuhowski have explored generic queue schedulers and have simulated a wide variety of queue scheduling methods [12]. A queue can actively request service based on spatial (queue length threshold) or temporal (timer) conditions. The queue server must solve the multiple request contention using proper scheduling schemes. Several forms of priority have been defined for the generic queue scheduler. Continuously gated priority (CGP) is essentially the head-of-line priority (HOLP). Queue gated priority (QGP) allows the queue currently being serviced to transmit up to its burst factor (BF) of cells. System gated priority (SGP) buffers all queue service requests into an arbiter and imposes an ordering on the requests. All the queues are allowed to transmit up to their burst factor of cells in that order. If the currently served queue runs out of cells, the next queue can transmit cells for a *work conservative* (WC) system and no other queue can transmit cells for a *non-work conservative* system.

Duan has proposed a 3-Dimensional-Queue (3DQ) that is used in an input-buffered ATM switch system [9]. The incoming cells are organized into multiple queues according to virtual connection, priority and destination. The memory pool is shared by all incoming cells and is grouped into fixed-size cell slots. The free list logic keeps track of all unused cell slots, assigns an available cell slot to each incoming cell, and collects cell slots after each cell has been delivered. 3DQ supports QoS at the per VC level. Every active VC has only one entry in the corresponding service queue. A non-empty VC can re-enter the service queue after being served. The scheduler selects the cell according to the “weight” of elements of a traffic matrix. The idea of combining the per-VC queueing, priority queueing and N-destination queueing is quite similar to the design proposed in this paper. But the selection of queue identifiers to build up the linked list and the scheduling of the outgoing cells are quite different. In addition, the 3DQ does not provide packet level discarding during overload periods.

The idea of *round-robin* scheduling is to circularly transmit one cell from each of the VCs with non-empty queues. Round-robin scheduling evenly distributes all the available link bandwidth to all active virtual circuits. If some virtual circuits do not need or cannot use as much as they are allowed, the remaining portion is equally distributed to all other VCs that can use it. Visiting the VCs in a round-robin fashion and forwarding one cell from each active VC upon each visit is, in principle, fairer than FIFO scheduling. However, in some cases, it may be necessary to allocate link bandwidth according to a set of weights. The *Weighted Round-Robin* (WRR) scheme proposed by Katevenis [11] visits the VCs with larger weights more frequently than those with smaller weights.

## 1.5. Thesis Overview

Chapter 2 describes the features of the Dynamic Queue Management chip. Chapter 3 gives the architecture of the Dynamic Queue Management chip and a series of operational examples. Chapter 4 elaborates the detailed design of each functional block, provides a detailed timing analysis. Chapter 5 describes several advanced algorithms used in the chip. Chapter 6 gives performance analysis and circuit complexity estimation. The last chapter summarizes the thesis.

# 2. FEATURES AND PRINCIPLES

The Dynamic Queue Management (DQM) chip allows a single 2.4 Gb/s OC-48 link or a combination of lower speed links such as 150 Mb/s OC-3, 622 Mb/s OC-12, and 1 Gb/s G-link to access high speed ATM switches such as the Washington University Gigabit Switch (WUGS) whose throughput is 2.4 Gb/s per port. The chip is designed to buffer cells received from the switch and forward them over one of the outgoing links. The major features of the DQM chip are listed below.

## 2.1. Dynamic Queue Assignment

The ATM cell format allows for as many as $2^{28}$ distinct virtual circuit connections on a single ATM link. Real switches implement only a small fraction of the full spectrum of possibilities, and often impose limitations on the choices of VPIs and VCIs. Many switches support only VP or only VC connections and those that only support VC connections usually restrict the VPI to be zero. In switches that support VC connections with different VPIs, it is generally necessary to configure the switches to specify which VCIs may be used with a given VPI.

The DQM implements per VC queueing using dynamic assignment, which allows the chip to support virtual path and virtual circuit connections with arbitrary choices of VPIs and VCIs and no explicit configuration of VCI ranges to particular VPIs. It uses set-associative lookup to assign queues to channels identified by the combination of a VPI and a VCI. This greatly simplifies the use of the chip and enables optimal use of the chip's per channel data structures.

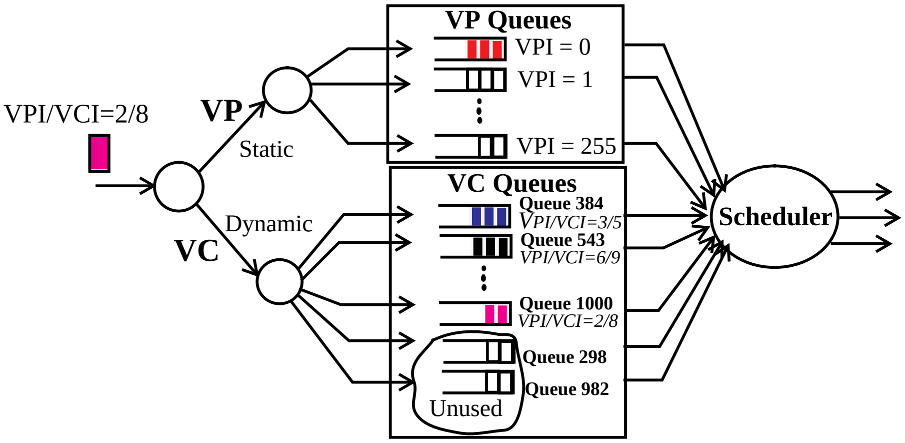


Figure 2-1  Dynamic Queue Assignment

Figure 2-1 illustrates the dynamic queue assignment mechanism. A static assignment is used for virtual path (VP) queues, since the number of distinct VPIs is fairly modest, 256 in this particular case. However, the number of distinct VPI/VCI combinations is far larger than the number of actual virtual circuits that a link will carry and larger than the number of distinct queues that can be managed economically. To avoid restricting the choice of VPIs and VCIs unduly, VC queues are assigned on a dynamic basis.

If an arriving cell belongs to a VP connection, the cell goes directly to the VP queue specified by its VPI. Otherwise, a set-associative lookup is performed to determine if there is an existing queue assigned to the same (VPI,VCI) pair as the arriving cell. If such a queue is found, the cell is appended to that queue. If there are no matching queues, an unused VC queue is assigned to that VC connection. The arriving cell is appended to the newly assigned queue.

## 2.2. Efficient Memory Management

Since bursty traffic is common in networks, large buffers are needed to accommodate temporary data surges. Arriving cells are first stored in the cell buffer before being forwarded to output links. The DQM chip allows all virtual circuit queues to share the entire cell buffer. The DQM chip is designed so that the cell buffer can be scaled up to very large sizes without increasing the chip complexity significantly.

The cell buffer is organized in units of *cell slots*. Queues are stored in the buffer as linked lists. A *Free Slot List* is maintained to keep track of all unused cell slots in the cell buffer. Both the cell buffer and all information to maintain the cell buffer (that is, all the links for the linked list queues and the free slot list) are stored in external memory. The only constraint that the DQM chip places on the buffer capacity is through the choice of pointers. With 20 bit pointers, the chip can support buffer sizes over 50 Mbytes, 24 bit pointers would allow for up to 800 Mbytes. For all practical purposes, the buffer capacity is not constrained by the DQM chip.

At gigabit speeds, the bandwidth of the external memory used by the DQM to store cells is a precious resource. A certain portion of this bandwidth must be used to manage the free slot list that is stored in the external memory, along with the waiting cells. The DQM chip incorporates an on-chip recycling cache that allows the free slot list to be maintained using only memory cycles that would otherwise go unused. This cache stores the location of a number of available cell slots.

Cell slots can usually be assigned to arriving cells from the cache and departing cells can usually return their cell slots to the cache, rather than accessing the off-chip free slot list. The off-chip list is only accessed to refresh or free up space in the cache. But these operations only needs to be performed when there is a mismatch in the rate at which cells arrive and depart. In this case, there are guaranteed to be unused memory cycles available. Therefore, with the internal recycling cache, the free slot list can be maintained in external memory with no additional memory bandwidth cost.

## 2.3. Efficient Weighted Fair Queueing

The DQM chip implements weighted round robin scheduling [11] using a novel approach we call the *Binary Scheduling Wheels* (BSW) algorithm [7]. The BSW algorithm implements multiple priorities at minimal cost, providing a wide range of rate options.

The BSW algorithm is a per VC based algorithm. Power of 2 weights can be assigned to individual virtual circuit connections. These weights determine the relative frequency with which cells are forwarded, allowing link bandwidth to be allocated appropriately during congestion periods. With 32 distinct weights, the BSW algorithm can assign bandwidth in amounts ranging from 2.4 Gb/s to less than one bit per second. Unlike naive implementations of weighted round-robin scheduling, the BSW algorithm interleaves cells from different channels as much as possible, minimizing the burstiness of the output data streams. The algorithm can be implemented in a very cost-effective way, requiring just a small increment in cost over a simple two priority level design. In order to be able to scale to large number of weights, the BSW algorithm incorporates a *Fast Forward* mechanism. It is well-suited to hardware implementation and allows cells to be scheduled and forwarded in essentially constant time.

## 2.4. Packet Level Discarding for Per VC Queues

To preserve packet integrity during overload, ATM switches often use packet level discard mechanisms such as Early Packet Discard [16][18][4], which were designed for use with FIFO queues. New algorithms are needed for per VC queueing, to minimize memory usage and preserve fairness and QoS properties of output scheduling algorithms. The DQM chip incorporates a new packet level discarding scheme for per VC queues, called the *Weighted Fair Goodput* (WFG) algorithm. The combination of WFG and BSW allows all virtual circuits to forward cells at reserved rates during overload periods and ensures that “well-behaved” virtual circuits (those that do not exceed their allocated rate) do not lose any data, and that data is discarded from “misbehaving” virtual circuits on a packet-by-packet basis, avoiding wasted link capacity during overload periods.

## 2.5. Flexible Link Configuration

The DQM chip implements a credit based link scheduler to allow the output links to be configured with excess capacity. It is possible to configure external links in such a way that total link bandwidth is greater than the bandwidth of the DQM chip. In particular, the DQM chip makes bandwidth that is not needed by one port, available to other ports. This allows links to send bursty traffic at higher rates if other links are not using their full bandwidth.

# 3. OVERALL DESIGN

## 3.1. Overview

To support the required output bandwidth of 2.4 Gb/s, the chip operates with an internal clock speed of 120 MHz. The internal cell time is the same as that of the OPP, which is 14 clock cycles. The data path between the OPP and the DQM chip is 32 bits. This allows cells to be received at a rate that is roughly 1.5 times the cell rate of an OC-48 link.

Figure 3-1 shows a block diagram of the DQM chip and its associated memory. The dash line illustrates the data flow from the switch fabric to the output links. ATM cells are received on a 32-bit wide interface, similar to the UTOPIA interface used for connecting ATM devices to SONET transmission circuits [1]. The DQM stores cells in the external memory and forwards them to one of possibly several output links.

There are nine functional blocks: the *Queue Selector*, the *Queue Manager*, the *Output Scheduler*, the *Free Slot Manager*, the *Cell Store*, the *Free Slot List*, the *Input Master*, the *Output Master*, and the *Memory Controller*. The *Queue Selector* dynamically assigns queues to virtual circuits. The *Queue Manager* maintains a list of all queues, keeping track of the first and the last cell in each queue. The *Output Scheduler* schedules the transmission of cells from various channels and allocates the chip's output bandwidth

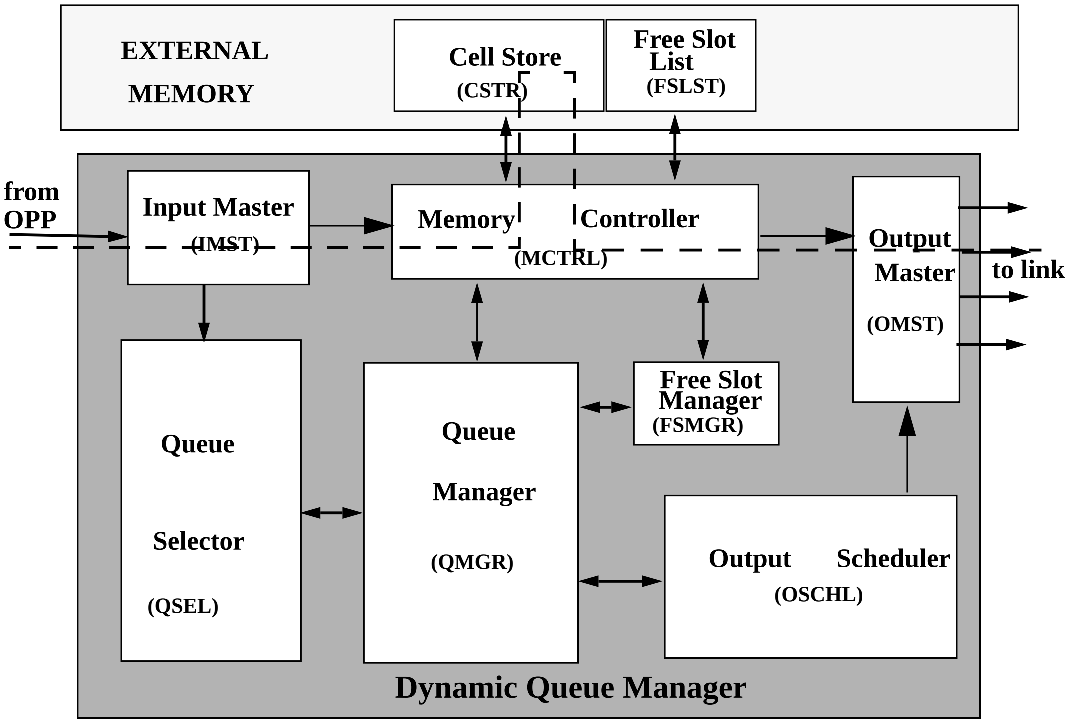


Figure 3-1 Block Diagram of the Dynamic Queue Management Chip

among different output links. The *Cell Store* buffers all incoming cells before transmission. The *Free Slot List* stores unused cell slots in the Cell Store and the *Free Slot Manager* maintains an on-chip cache and manages the Free Slot List. The *Input Master* receives cells from the switch and retrieves control information. The *Memory Controller* interfaces to the external memory and handles the necessary format conversions, needed to map cells into memory. The *Output Master* forwards cells to the output links.

### 3.1.1. Queue Selector (QSEL)

The Queue Selector (QSEL) maps the VPI and VCI fields of incoming cells to dynamically assigned queue identifiers. It contains a *Queue Lookup Table* (QLT), which operates like a *Content-Addressable Memory* (CAM). Logically, one can think of the QLT as a set of entries, each containing a (VPI,VCI) and a queue identifier. When a cell arrives, the VPI and VCI fields of the cell are compared to the stored entries. If a matching entry is found, the queue identifier in the entry gives the number of the queue that the cell is to be appended to. If there is no matching entry, a free queue is allocated and a new entry is created, and initialized with the VPI and VCI of the incoming cell and the identifier of the queue just allocated.

Figure 3-2 illustrates the Queue Selector data structures. In addition to the QLT, the Queue Selector contains a *Free Queue List* and an *Address Map*. Queues are allocated from the Free Queue List as needed. When the transmission of a cell by the Queue Manager causes a queue to become empty, the identifier for that queue is returned to the Queue Selector, which adds it back into the Free Queue List. The Address Map specifies the QLT entry where a specified queue identifier is stored. It is used to remove an entry from the QLT when a queue becomes empty. Specifically, whenever a queue identifier is returned to the Free Queue List, the Queue Selector uses the queue identifier to select an entry from the Address Map. The value returned is then used to deallocate the specified QLT entry.

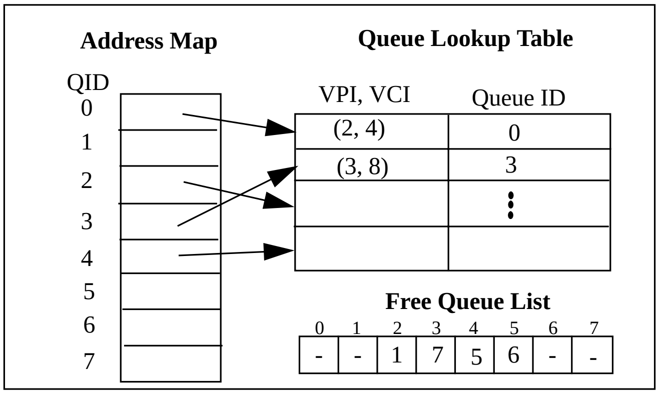


Figure 3-2 Queue Selector Data Structures

To obtain the most cost-effective implementation, the QLT is implemented using a *Set-Associative Memory* (SAM). SAMs can be implemented with conventional SRAM and some auxiliary logic, making them a good deal cheaper than CAMs. Given a VPI and a

VCI, the set-associative memory in the Queue Selector returns a set of entries, any one of which could be used for storing information relating to that VPI and VCI combination. A set of entries is selected using a subset of the bits of the VPI and VCI. Each entry contains a valid bit, a tag and a queue identifier. The tag is formed from the bits of the VPI and VCI not used to select the set.

When a cell is received by the DQM, its VPI and VCI are passed to the Queue Selector, which retrieves a set of entries from the set-associative memory. The tag of the incoming cell is compared to the tag fields of all entries of the set in parallel. If the tag field of some valid entry in the set matches the tag of the incoming cell, the queue identifier stored in that entry identifies the queue that the cell should be appended to. If there is no tag match, it means that no cell belonging to this combination of VPI/VCI is currently stored in the DQM's memory, and so an unused queue identifier should be assigned. To perform this assignment, the Queue Selector picks an unused entry from the set returned by the set-associative memory and obtains an unused queue identifier from the list of free queue identifiers that it maintains. It then copies the tag of the cell and the number of the allocated queue into the selected entry, sets its valid bit and writes it back to the set associative memory.

When the DQM transmits the last cell from some queue, the number of that queue is passed to the Queue Selector which returns the queue to its list of available queues and clears the valid bit in the corresponding entry of the set-associative memory. Thus, both queues and entries in the set-associative memory are used only for those connections for which the DQM is storing cells.

When a cell is received by the DQM, it is possible that the set-associative lookup will yield a set of entries, all of which are in use (have the valid bit set) and none of which have matching tags. In this case, the cell must be discarded. This is referred to as overflow. We can make overflow less frequent by augmenting the set associative memory with a small Content Addressable Memory (CAM). The entire VPI and VCI are used as the key field for the CAM. In the value field, we store the queue identifier that is assigned to the virtual circuit. Figure 3-3 shows the set-associative lookup with CAM.

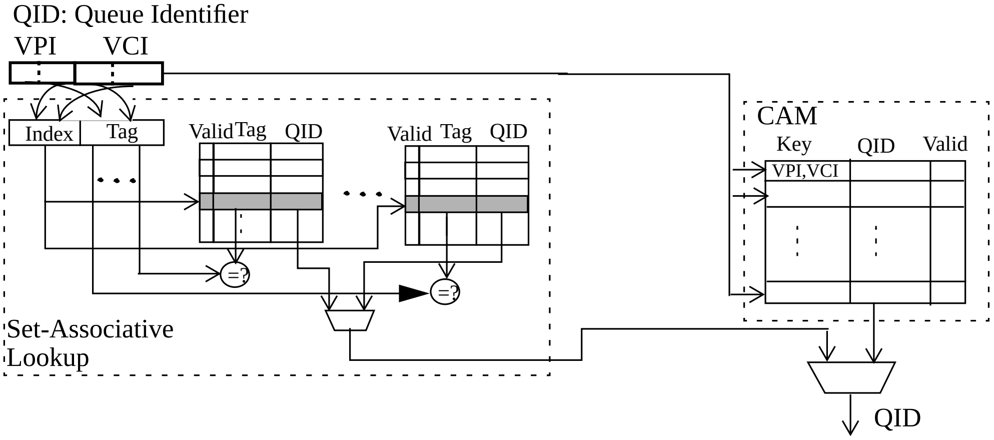


Figure 3-3 Set-Associative Lookup with CAM

When a cell enters the DQM, a CAM lookup is performed in parallel with the set-associative lookup described earlier. There are several cases that can then arise: If the set returned by the set-associative memory has a matching entry, it is used as previously described. If the set has no matching entry, but the CAM contains a matching entry, then the queue identified in the matching CAM entry is used for the arriving cell. In this case, if the set returned from the set-associative memory has an unused entry, the information in the CAM entry is transferred from the CAM entry to the entry in the set-associative memory, freeing up the CAM entry. If neither the set returned by the set-associative memory, nor the CAM has a matching entry, then a new queue is allocated. If the set has one or more free entries, one of them is allocated. Otherwise an entry in the CAM is allocated.

### 3.1.2. Queue Manager (QMGR)

The Queue Manager (QMGR) maintains a queue list for every possible virtual path queue and virtual circuit queue by keeping pointers to the head and tail of the queues. The actual queues are organized as linked lists and are stored in the external memory (Cell Store). Each cell slot in the Cell Store contains one cell and a pointer to the next cell in the queue. Figure 3-4 shows the queues maintained by the QMGR for a scaled-down configuration.

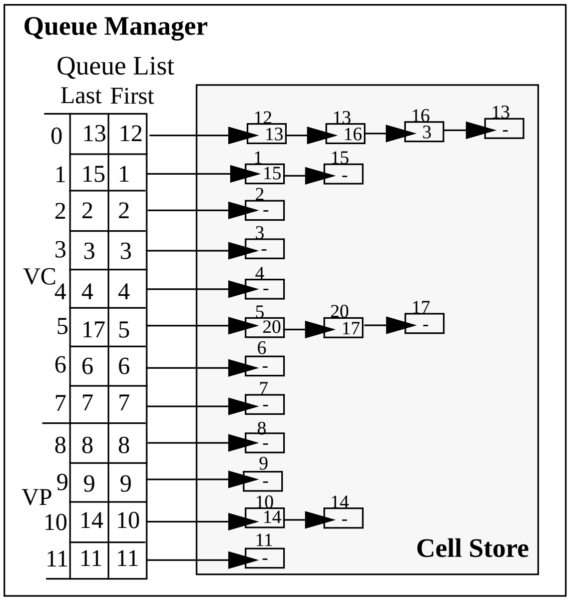


Figure 3-4 Queues Maintained by the Queue Manager

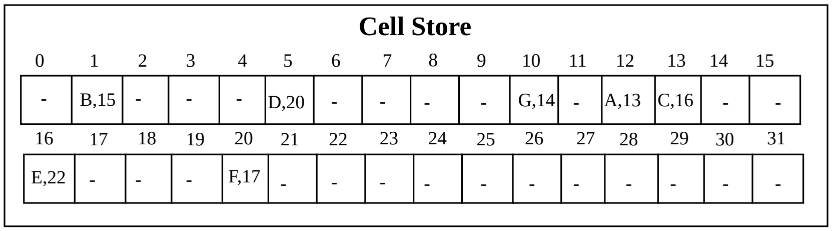
The queue identifiers generated by the Queue Selector are used to index the queue list. Each entry in the queue list contains pointers to the first and last cell of the queue. In order to simplify the design, the LAST pointer always points to an empty cell slot in the Cell Store which can accept the next arriving cell.

For an arriving cell, the QMGR informs the Cell Store to store the cell in the cell slot which the last cell in the queue points to. The QMGR also obtains a free slot from the Free Slot Manager and updates its queue list. The free slot number is written into the

Cell Store along with the cell. For an outgoing cell, the QMGR obtains a queue identifier of the queue to output a cell from the Output Scheduler. The QMGR sends the cell slot number of the first cell in the queue to the Cell Store. The pointer stored with the departing cell is used to update the queue list. The cell slot that contains the departing cell is then returned to the Free Slot Manager. The QMGR notifies the Queue Selector and the Output Scheduler to remove the queue if the first and last pointers in the queue both refer to the empty cell slot after modifying the queue list.

## 3.1.3. Cell Store (CSTR)

The Cell Store (CSTR) stores cells before they are forwarded to the external links. Cells are organized as linked lists on a per connection basis as shown in Figure 3-4. The minimum units in the CSTR are called cell slots and are indexed by slot numbers. Each cell slot can accommodate one cell and a pointer to build up the linked list. Figure 3-5 illustrates the cell slots in the CSTR, where the letter designates a stored cell and the number is the pointer to the next cell in the queue.



**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | B,15 | - | - | - | D,20 | - | - | - | - | G,14 | - | A,13 | C,16 | - | - |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E,22 | - | - | - | F,17 | - | - | - | - | - | - | - | - | - | - | - |

Figure 3-5  Cell Slots in the Cell Store

For an arriving cell, the CSTR receives a slot number and a pointer from the Queue Manager (QMGR). It writes the entire cell and the pointer to the cell slot indicated by the slot number. For an departing cell, the CSTR receives a slot number from QMGR, and reads the cell and the pointer out of the cell slot. It forwards the cell to the Output Master (OMST), and forwards the pointer to the QMGR.

### 3.1.4. Output Scheduler (OSCHL)

The Output Scheduler (OSCHL) determines the order in which different non-empty queues are selected for transmission. Because the chip is designed to support multiple outputs (up to 16 OC-3 interfaces, in some configurations), the Output Scheduler has separate scheduling data structures for each output. There are two scheduling mechanisms that the Output Scheduler can choose from. One is two priority round-robin scheduling. The other is weighted round-robin scheduling. The basic design implements two priority scheduling scheme. An efficient implementation of weighted round-robin scheduling is discussed in detail in Chapter 5.

In the two priority case, each output has a high priority list and a low priority list. Queues on these lists contain at least one cell. If a queue becomes empty after forwarding a cell, it is removed from the list. Output links are served in a round-robin fashion. The idea of round-robin scheduling is that during each round, everyone is visited exactly once. Each of the 16 outputs is able to forward one cell every 16 cell times. When an output is selected, the high priority list on that output is served before the low priority list. The low priority list can be processed only if the high priority list is empty. Queues on the lists are also served in a round-robin fashion. In this way, real-time streams can avoid being delayed by bursty data traffic.

The OSCHL uses the queue identifiers maintained by the Queue Manager to identify different active queues. Figure 3-6 shows a scaled-down version of the high and low priority lists that the OSCHL maintains. The queue identifiers of all non-empty queues are linked together to form circular lists on the priority lists.

The OSCHL sends the queue identifier of the scheduled queue to the Queue Manager and sends the corresponding output link number to the Output Master. For a new connection, it inserts a queue identifier in the proper place in its list according to the

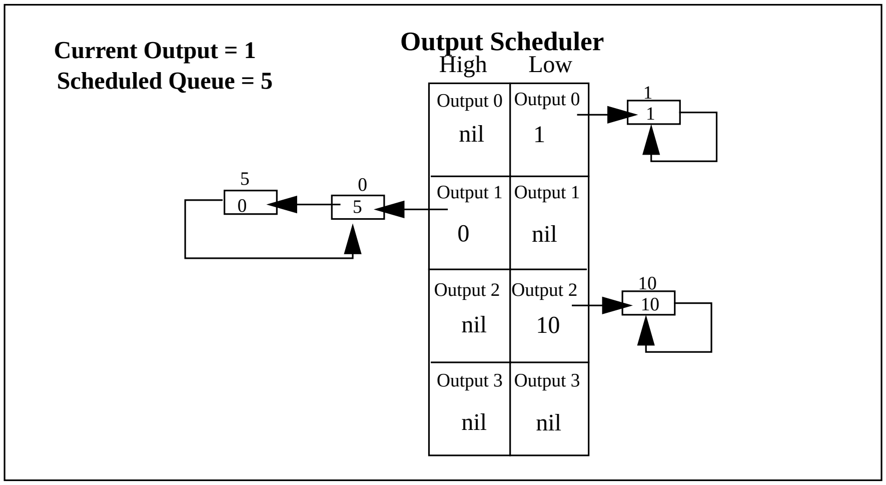


Figure 3-6 High/Low Priority Lists in Output Scheduler

priority bit and the output link number. It removes a queue identifier from the list when the queue is empty, as informed by the QMGR.

If weighed round-robin scheduling is used, each queue is assigned a weight from a fixed set of power of 2 weights. For each output, the OSCHL maintains an array of scheduling lists, with one list for each weight. This is illustrated in Figure 3-7, which shows a scaled-down version of the OSCHL data structures, configured for four outputs and four weights. The actual chip supports up to 16 outputs and 32 weights.

The scheduling lists are organized cyclically and are referred to as scheduling wheels. The weights for the different scheduling wheels determine the relative frequency with which the queues are scheduled. Queues on the weight 0 wheel are visited twice as often as queues on the weight 1 wheel, four times as often as queues on the weight 2 wheel, and so forth. The scheduling algorithm used by the OSCHL is called the *Binary Scheduling Wheels* algorithm and is described in detail in Chapter 5.

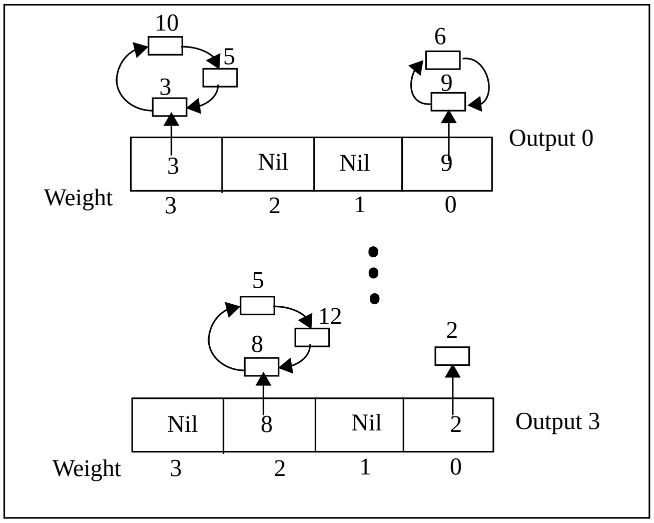
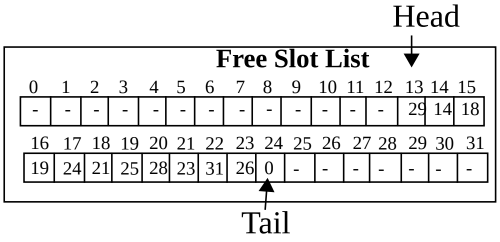


Figure 3-7  Scheduling Wheels in the Output Scheduler

### 3.1.5.  Free Slot Manager (FSMGR) and Free Slot List (FSLST)

The Free Slot List (FSLST) stores the currently unused slot numbers in the Cell Store. Because of the size of the FSLST, it is stored in the external memory. The FSLST is implemented as a circular list with a head pointer and a tail pointer. Figure 3-8 shows a scaled-down version of the Free Slot List.



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | - | - | 29 | 14 | 18 |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 24 | 21 | 25 | 28 | 23 | 31 | 26 | 0 | - | - | - | - | - | - | - |

Figure 3-8  Free Slot List

The Free Slot Manager (FSMGR) manages the Free Slot List. It keeps an on-chip free slot cache. The FSMGR assigns an empty cell slot from its cache to an arriving cell. After a cell departs, the corresponding cell slot is returned to the free slot cache. If there are too many free slots in the cache when there is no arriving cell for a long time, the FSMGR moves free slots to the Free Slot List using an idle memory write cycle. If there is no departing cell for a long time, the free slot cache will run short of free slots. The FSMGR then reads free slots from the Free Slot List using an idle memory read cycle.

### 3.1.6. Input Master (IMST)

The Input Master (IMST) receives cells from the Output Port Processor (OPP). It retrieves control information from the cell. The IMST sends the VPI, VCI, type, PT, link number and priority of the incoming cell to the Queue Selector. It forwards the entire cell to the Memory Controller.

### 3.1.7. Output Master (OMST)

The Output Master (OMST) forwards cells to the output links. It receives the departing cell from the Memory Controller and the output link that the cell is destined for from the Output Scheduler. The OMST provides separate buffers for each of the chip’s four UTOPIA interfaces as shown in Figure 3-9.

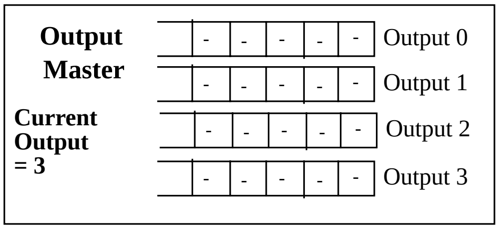


Figure 3-9 Buffers in the Output Master

### 3.1.8. Memory Controller (MCTRL)

The *Memory Controller* (MCTRL) controls the external memory blocks: Cell Store and the Free Slot List. It communicates with the Input Master, the Output Master, the Queue Manager and the Free Slot Manager. The Memory Controller also handles cell format conversion and memory address translation.

## 3.2. Operational Examples

The following examples illustrate the operation of the major blocks. More examples are given in Appendix B.

In order to simplify the examples, we assume a 2-bit VPI, a 3-bit VCI, a set-associative memory with 8 sets and 2 entries each, 32 slots in the Cell Store and 4 output links. We allow 8 VC connections, 4 VP connections. Cells are denoted as A, B...Z. Each incoming cell provides its VPI, VCI, type, priority, and the output number. Type can be either VC connection or VP connection and priority can be high or low.

Table 3-1 shows the incoming cells for a few cell times after the initialization. We will go over the operations of the first few cell times and leave the rest for Appendix B.

Figure 3-10 gives the overall timing of the cell arrival and departure. "A-1" denotes that cell A is destined for output 1. Each cell time is divided into a write cycle followed by a read cycle. During the write cycle, denoted as “W”, the arriving cell is written into the Cell Store, while during the read cycle, a cell is read out of the Cell Store. "R1" means the current read cycle is for a cell destined for output 1.

**Table 3-1 Arriving Cell Information**

| Arrival Time | Cell Content | VPI/VCI | Type | Priority | Output Number | Queue ID |
|---|---|---|---|---|---|---|
| 0 | A | 0/2 | VC | HIGH | 1 | 0 |
| 1 | B | 0/2 | VC | HIGH | 1 | 0 |
| 2 | C | 2/3 | VP | LOW | 2 | 10 |
| 3 | D | 3/5 | VC | LOW | 0 | 1 |
| 4 | E | 0/2 | VC | HIGH | 1 | 0 |
| 5 | F | 3/5 | VC | LOW | 0 | 1 |
| 6 | G | 2/5 | VP | LOW | 2 | 10 |
| 7 | H | 1/6 | VC | HIGH | 1 | 2 |
| 8 | - | - | - | - | - | - |
| 9 | I | 3/7 | VP | LOW | 2 | 11 |
| 10 | - | - | - | - | - | - |
| 11 | - | - | - | - | - | - |

Because all the output links are served in round-robin fashion, each output can only send one cell per round. For example, Cell A destined for output 1 arrives at Cell Time 0, and leaves at Cell Time 5 when the read cycle is for output 1. Also, queues on each output are also served in round-robin fashion with those on the high priority lists served before those on the low priority lists.The following figures will explain how decisions are made. If there is no queue on a particular output during its read cycle, the read cycle for that output is not used.

Figure 3-11 illustrates the states at the end of Cell Time 0. During Cell Time 0, cell *A* comes in with VPI=0, VCI=2, TYPE=VC, PRI=HIGH, OUT=1. Since it is a VC connection, a queue identifier is dynamically assigned to it. The Queue Selector uses the lower 1 bit of the VPI and lower 2 bits of the VCI to form an address (ADDR). A TAG field is formed by the highest 1 bit of the VPI and highest 1 bit of the VCI. The Queue

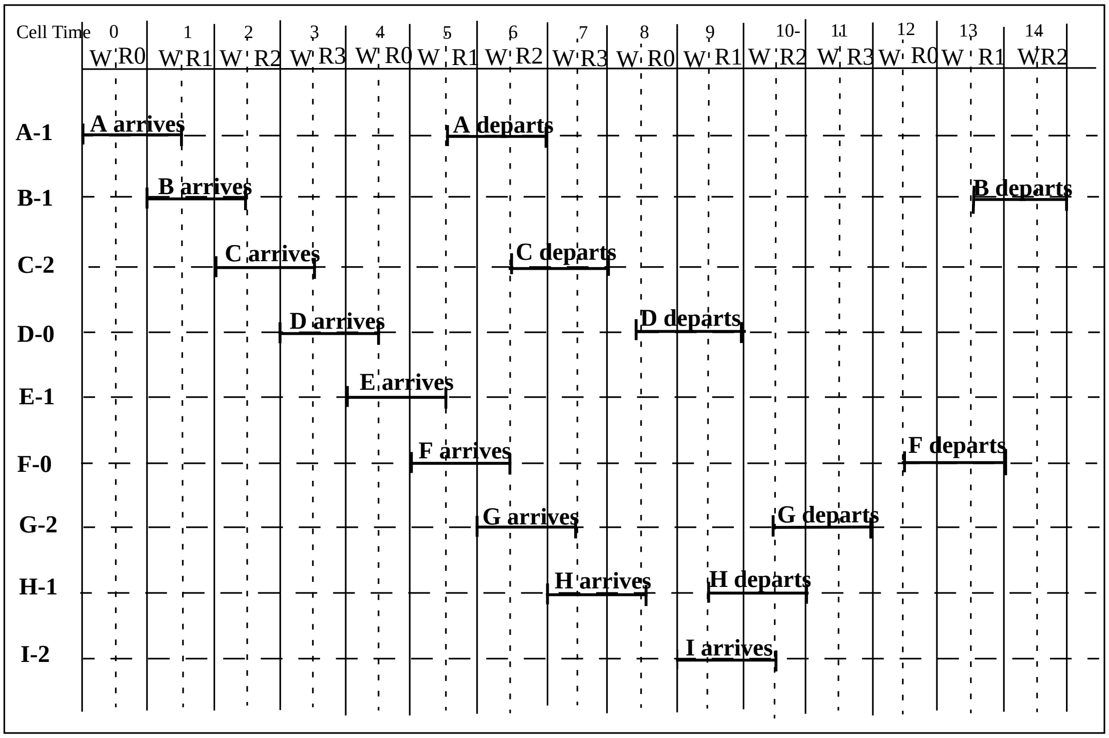
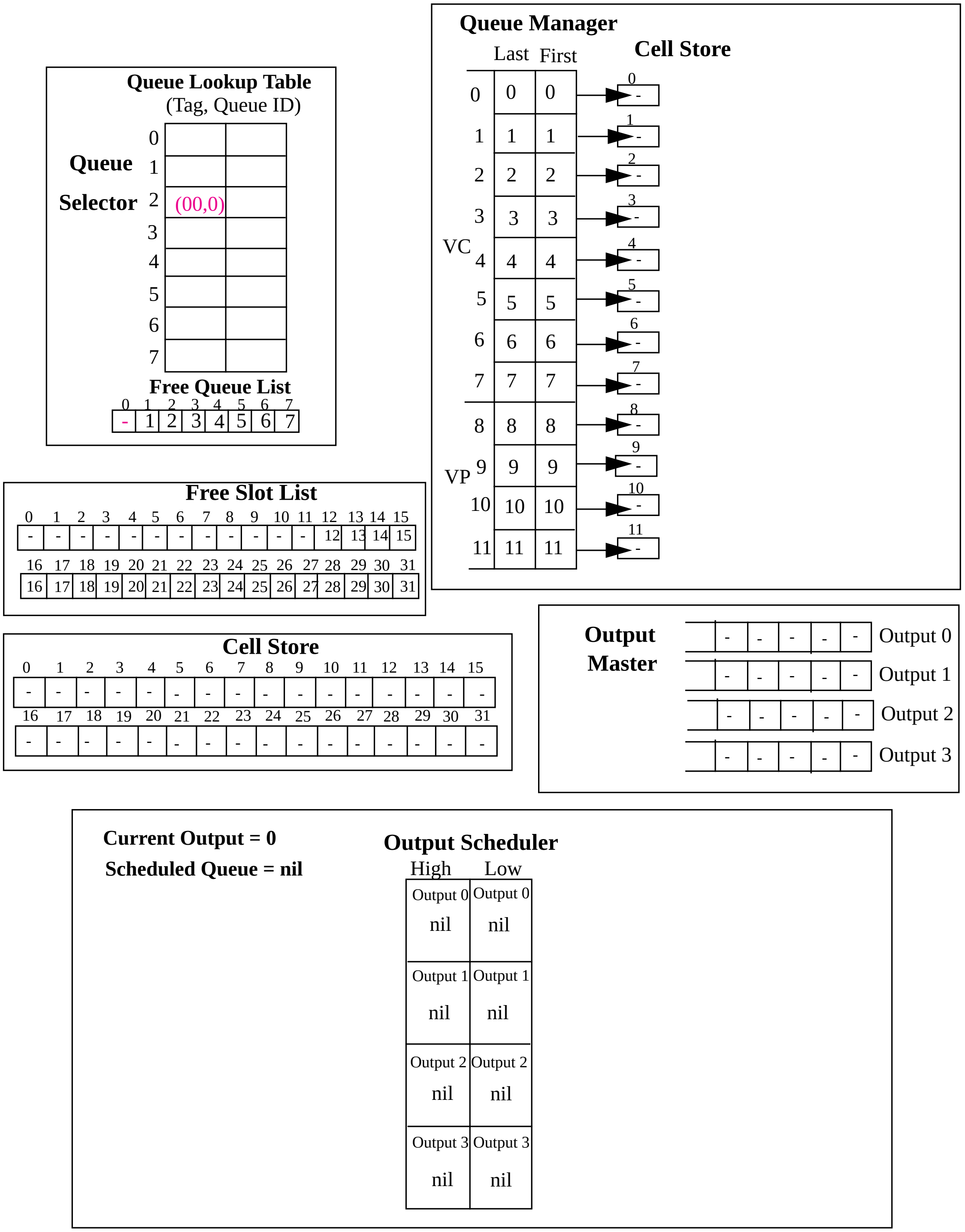


Figure 3-10 Timing of Cell Arrival and Departure

Selector uses the address to retrieve a set of entries from its set-associative memory and match them against the TAG field. If there is a matched tag, the Queue Selector will return the queue identifier stored in the matching entry. If not, there is no queue currently assigned to the connection and the first available queue identifier in the Free Queue List will be assigned to the connection and stored in the first unused entry of the set, together with the TAG field. In this case, ADDR=2, TAG=$(00)_2$. Because there is no tag match, the Queue Selector assigns 0 as the queue identifier to it. So the (TAG, QID) pair (00,0) is written to address 2, entry 0. The next available queue identifier in the Free Queue List is now 1. The Queue Selector forwards the QID=0 to the Queue Manager. The current output in the Output Scheduler is 0. There is no queue on either the high priority list or the low priority list, so no cell is scheduled to the Output Buffer. No other changes occur in this cell time.

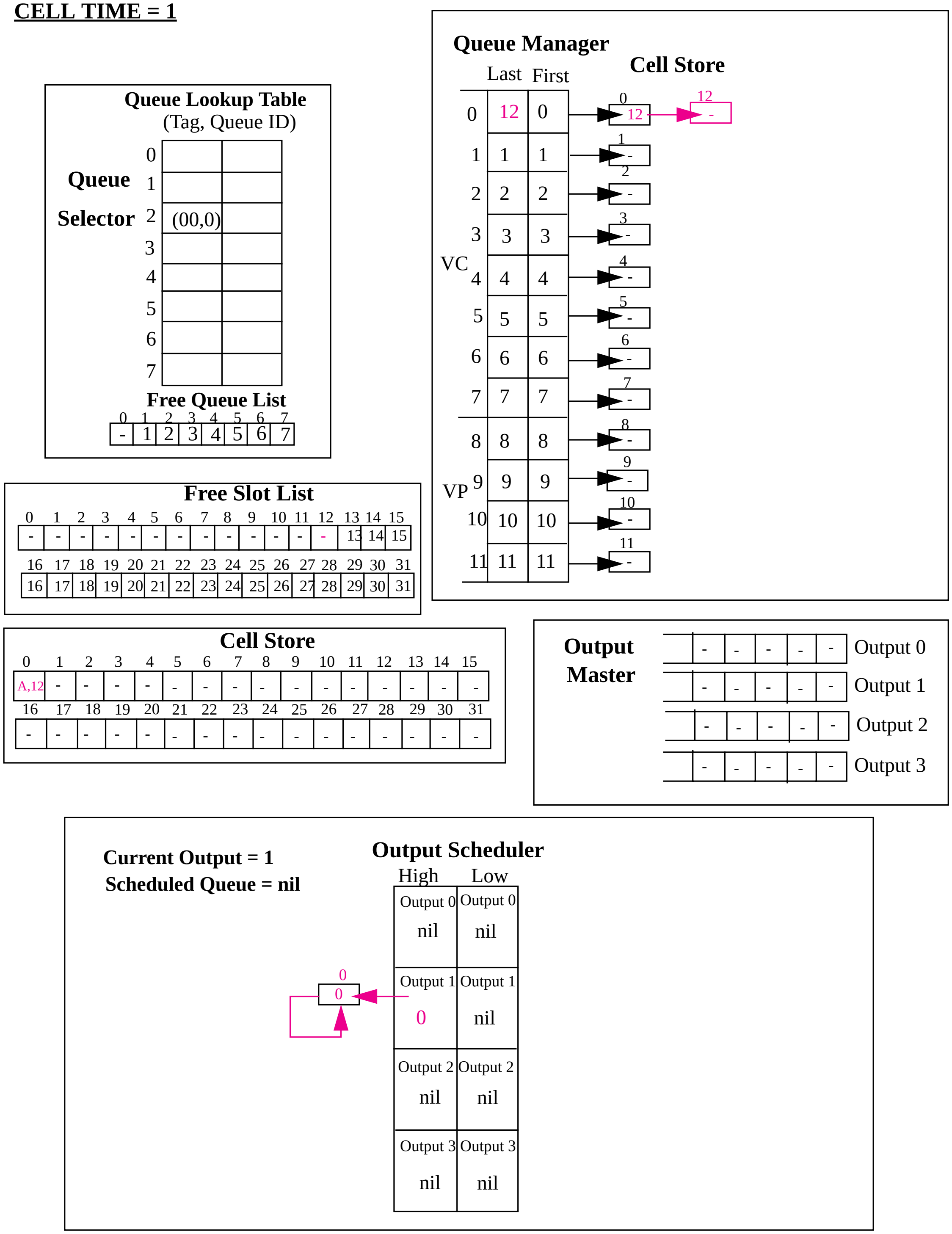
**CELL TIME = 0**



* Cell A, VC, Index 2, Tag 00, VC Queue 0

Figure 3-11 Status at the End of Cell Time 0

**CELL TIME = 1**



| Queue Selector | Queue Lookup Table (Tag, Queue ID) | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

**Free Queue List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

**Free Slot List**

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

**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A,12 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Queue Manager**

| | | Last | First | Cell Store |
|---|---|---|---|---|
| VC | 0 | 12 | 0 | 12 |
| | 1 | 1 | 1 | - |
| | 2 | 2 | 2 | - |
| | 3 | 3 | 3 | - |
| | 4 | 4 | 4 | - |
| | 5 | 5 | 5 | - |
| | 6 | 6 | 6 | - |
| | 7 | 7 | 7 | - |
| VP | 8 | 8 | 8 | - |
| | 9 | 9 | 9 | - |
| | 10 | 10 | 10 | - |
| | 11 | 11 | 11 | - |

**Output Master**

| | | | | | | |
|---|---|---|---|---|---|---|
| | - | - | - | - | - | Output 0 |
| | - | - | - | - | - | Output 1 |
| | - | - | - | - | - | Output 2 |
| | - | - | - | - | - | Output 3 |

**Output Scheduler**

| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>nil |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>nil |
| Output 3<br>nil | Output 3<br>nil |

* Cell B, VC, Index 010, Tag 00, Queue ID 0

* Queue 0 is added to Output 1, High Priority, Cell A is written into Slot 0

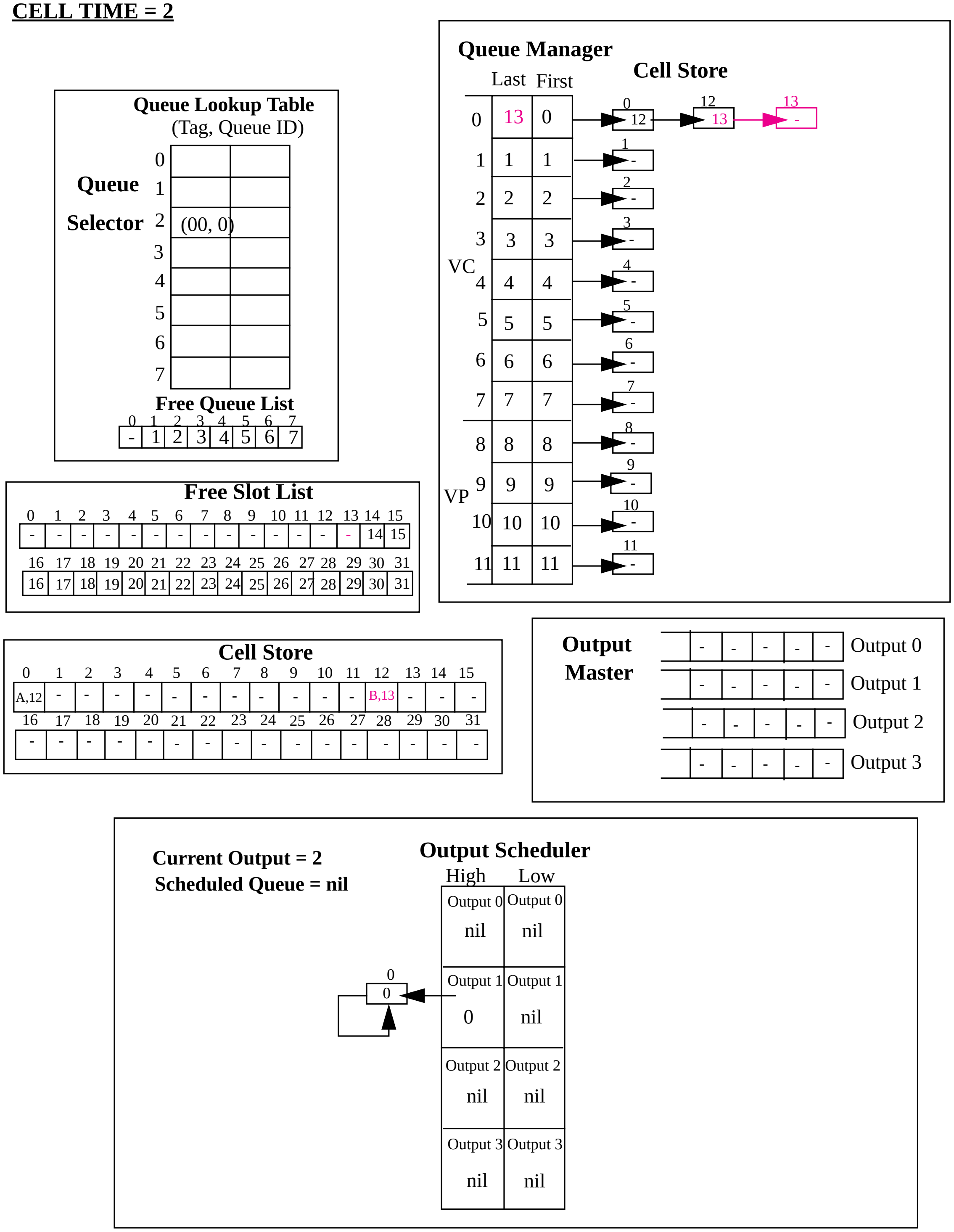
Figure 3-12 Status at the End of Cell Time 1

Figure 3-12 shows the status at the end of Cell Time 1. During Cell Time 1, the Queue Manager uses the QID as the index to get the value in the LAST field, which indicates the cell slot to store the arriving cell. Since LAST=0, Cell *A* is written to Slot 0 in the Cell Store. The Queue Manager asks for another free slot from the Free Slot List to update the LAST field. The number of this slot is also written into the Cell Store with the cell, as the pointer (PTR) field. At the end of Cell Time 1, LAST=12 and the next available free slot in the Free Slot List is 13. Since PRI=HIGH and OUT=1, and it is a new connection, the Output Scheduler adds the queue to the high priority list on output 1. Besides the operations on Cell A, Cell B comes in with VPI=0, VCI=2, TYPE=VC, PRI=HIGH, OUT=1. The Queue Selector computes the ADDR=2, TAG=(00)2. It checks the TAG field in its entry and finds a tag match. So it forwards the QID=0 to the Queue Manager. The current output in the Output Scheduler is 1. Because at the beginning of the cell time, the output list is empty, no cell output occurs in this cell time.

Figure 3-13 shows the status at the end of Cell Time 2. During Cell Time 2, Cell *B* is written into the slot 12 in the Cell Store indicated by the LAST field of QID 0 with the PTR=13. The LAST field is also updated to 13 and the first element in the Free Slot List is moved to 14. Cell C comes in with VPI=2, VCI=3, TYPE=VP, PRI=LOW, OUT=2. It is a VP connection, so the Queue Selector statically assigns QID=VP+8=10 to the connection. No other changes occur.

Figure 3-14 shows the status at the end of Cell Time 3. During Cell Time 3, since the LAST field of queue 10 points to cell slot 10, Cell *C* goes into slot 10 in the Cell Store. The LAST field is changed to 14 and the header of the Free Slot List is advanced to point to 15. Since it is a new connection, queue 10 is added to the low priority list on output 2 in the Output Scheduler. Cell *D* comes with VPI=3, VCI=5, TYPE=VC, PRI=LOW, OUT=0. The index to the Queue Selector is computed as 5 and TAG is $(11)_2$. There is no tag match in this case so a new QID=1 is assigned to it.

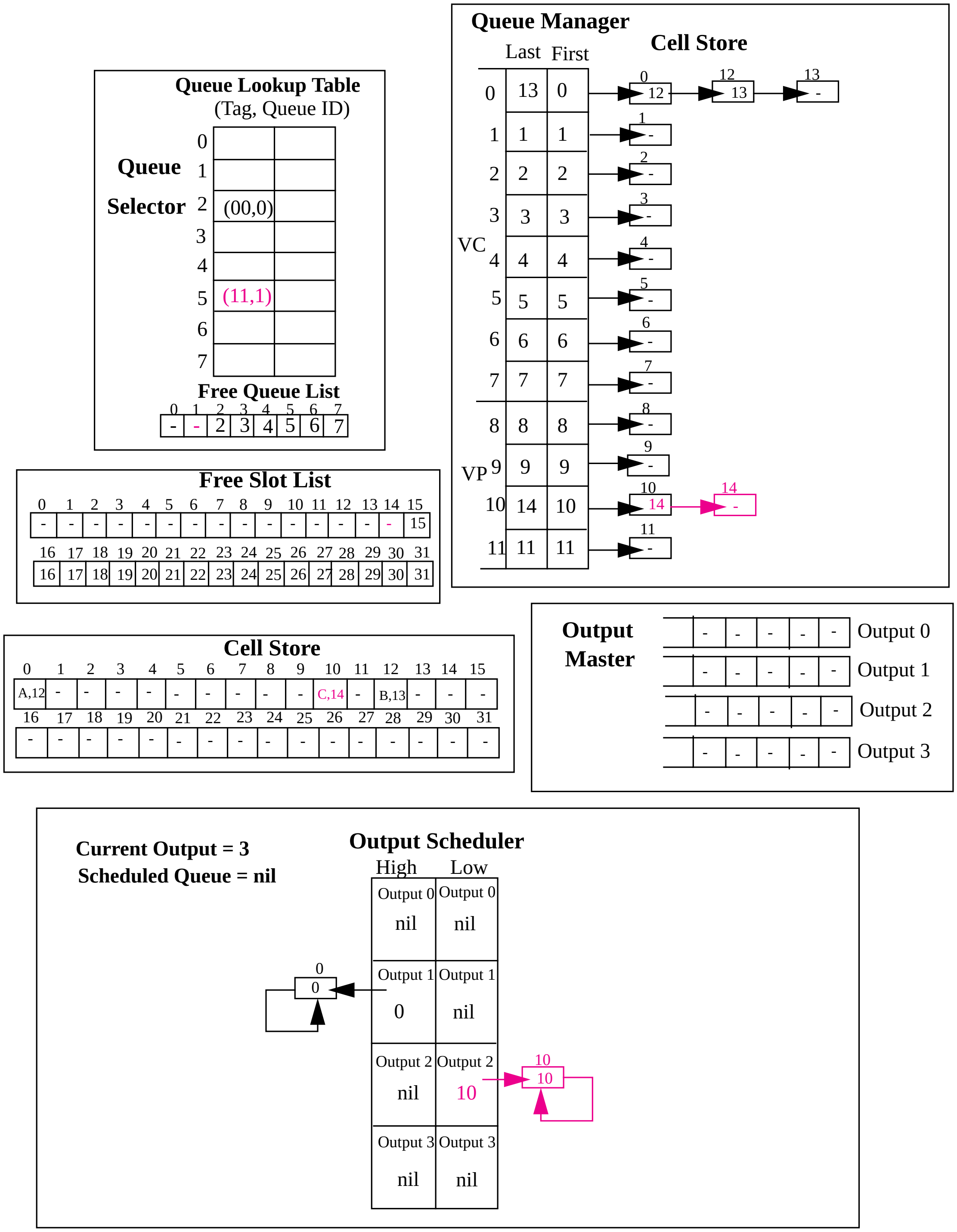
**CELL TIME = 2**



* Cell C, VP, Queue ID 10
* Cell B is written into Slot 12

Figure 3-13 Status at the End of Cell Time 2

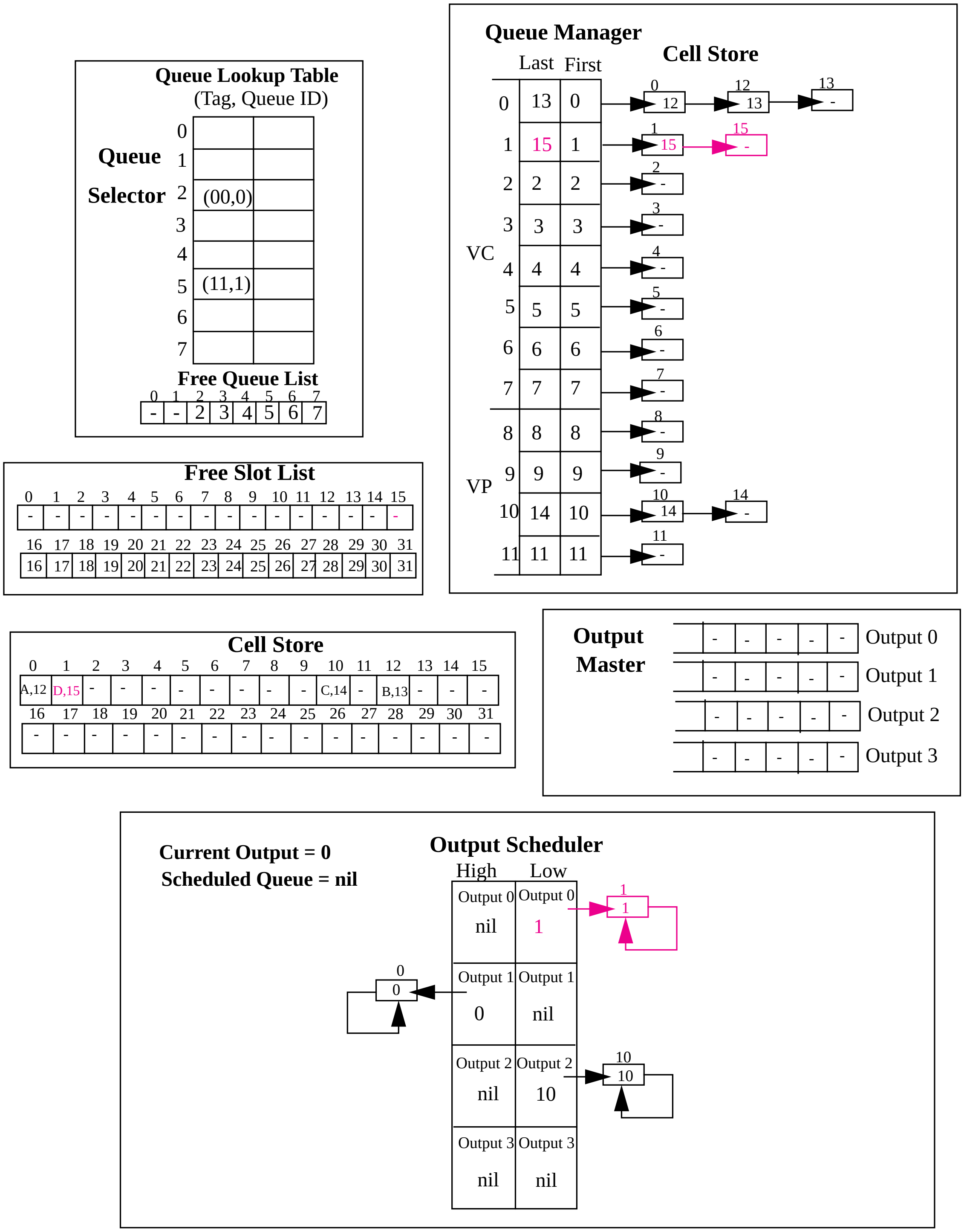
**CELL TIME = 3**



* Cell D, VC, Index 5, Tag 11, Queue ID 1
* Queue 10 is added to Output 2, Low Priority, Cell C is written into Slot 10,

Figure 3-14 Status at the End of Cell Time 3

**CELL TIME = 4**



* Cell E, VC, Index 2, Tag 00, Queue ID 0

* Queue 1 is added to Output 0, Low Priority, Cell D is written into Slot 1,

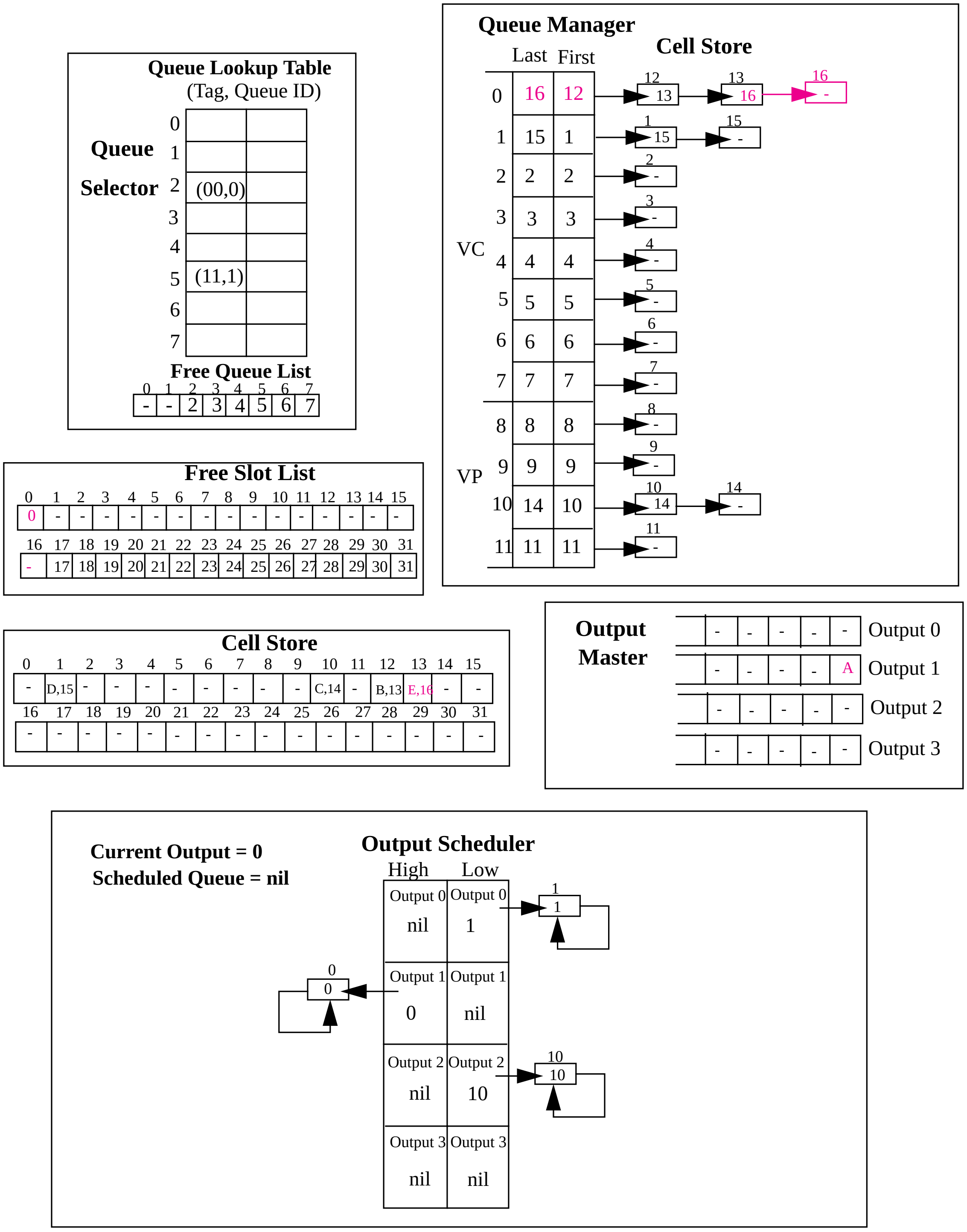
Figure 3-15 Status at the End of Cell Time 4

During Cell Time 4, as shown in Figure 3-15, Queue 1 is attached to the low priority list on output 0 in the Output Scheduler. Cell *D* is written into Slot 1 in the Cell Store. Cell *E* comes in with VPI=0, VCI=2, TYPE=VC, PRI=HIGH, OUT=1. It belongs to the same virtual circuit connection as Cells *A* and *B* and is assigned QID 0.

Figure 3-16 illustrates the status at the end of Cell Time 5. During Cell Time 5, Cell *E* goes to slot 13. Cell *F* arrives with VPI=3, VCI=5, TYPE=VC, PRI=LOW, OUT=0. The major difference between this cell time and previous ones is that Queue 0 is scheduled by the Output Scheduler to output its cell to output 1. Since the first cell of that queue is Cell *A*, Cell *A* is read out from the Cell Store and written to the Output Master. Slot 0 is returned to the Free Slot List.

The operations in later cell times are similar, so detailed descriptions are omitted. However, figures for the next 5 steps are included in appendix B.

**CELL TIME = 5**



Queue Lookup Table
(Tag, Queue ID)

| Queue Selector | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

Free Queue List

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | - | 2 | 3 | 4 | 5 | 6 | 7 |

Queue Manager

| | Last | First |
|---|---|---|
| 0 | 16 | 12 |
| 1 | 15 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 4 | 4 |
| 5 | 5 | 5 |
| 6 | 6 | 6 |
| 7 | 7 | 7 |
| 8 | 8 | 8 |
| 9 | 9 | 9 |
| 10 | 14 | 10 |
| 11 | 11 | 11 |

Free Slot List

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

Cell Store

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | D,15 | - | - | - | - | - | - | - | - | C,14 | - | B,13 | E,16 | - | - |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

Output Master

| | | | | | |
|---|---|---|---|---|---|
| - | - | - | - | - | Output 0 |
| - | - | - | - | A | Output 1 |
| - | - | - | - | - | Output 2 |
| - | - | - | - | - | Output 3 |

**Current Output = 0**
**Scheduled Queue = nil**

Output Scheduler

| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>1 |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>10 |
| Output 3<br>nil | Output 3<br>nil |

* Cell F, VC, Index 5, Tag 11, Queue ID 1

* Cell E is written into Slot 13* Queue 0 is selected, Cell A is send to OMST, return slot 0

Figure 3-16 Status at the End of Cell Time 5

# 4. DETAILED DESIGN

## 4.1. External Interfaces

### 4.1.1. Output Port Processor (OPP)

The Output Port Processor (OPP) is part of the Washington University Gigabit Switch (WUGS). With 32-bit data paths clocked at a rate of 120 MHz, the system is fast enough for external links up to 2.4 Gb/s.

The Dynamic Queue Management (DQM) Chip receives cells from the OPP in the format shown in Figure 4-1. The various header fields have the following interpretation.

**GFC - Generic Flow Control:** This 4-bit field is used for carrying local flow control information between an end host and the first ATM switching node to which it is directly connected (it is not carried through end-to-end). This field is ignored in the WUGS design.

**VPI - Virtual Path Identifier:** The ATM standard supports two types of connections. In virtual path connections, the VCI field is preserved end-to-end, and only the VPI field

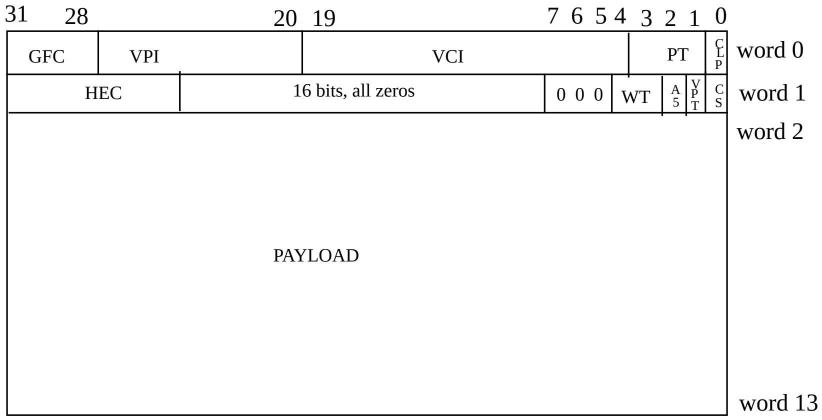


Figure 4-1 Cell Format

contents are used by the network to route cells. On the other hand, in virtual circuit connections, both the VCI and the VPI fields are used to route cells, and so neither is preserved end-to-end.

**VCI - Virtual Circuit Identifier:** As mentioned earlier, the VCI field is used along with VPI for routing cells in a virtual circuit connection. In virtual path connections, it is preserved end-to-end, and can therefore be used by end hosts to demultiplex different cell streams routed on the same virtual path.

**CLP - Cell Loss Priority:** This 1-bit field is used to indicate low priority cells. The source may set this bit to 0 or 1. If the bit is 1, switches along the connection path know that the cell is of low priority, and preferentially discard such cells if the network is encountering congestion. Because the CLP bit can change from cell to cell in the same virtual circuit, the DQM chip uses the CS field instead to indicate high/low priority.

**PT - Payload Type:** This 3-bit field, along with some of the other fields in the ATM cell header, determines the cell type.

**HEC - Header Error Check:** The header error check is an 8-bit CRC that is computed only over the header fields. The CRC computation is based on the polynomial: $x^8 + x^2 + x + 1$.

**CS - Continuous Stream:** Continuous media connections are those in which the data rate is either constant, or has low variance over time (e.g. compressed video and voice). Discrete media connections have higher variance in their data rates, and are often described as “bursty”. The CS bit field has a value of 1 if the connection carries continuous media traffic, and 0 if it carries discrete media. Continuous streams gets priority over discrete streams. The DQM chip use this field to distinguish high/low priorities.

**VPT - Virtual Path Termination:** This 1-bit field is 1 if the cell is part of a virtual circuit connection, 0 if it is part of a virtual path connection. Virtual path connections and virtual circuit connections are treated differently in the DQM chip for queue identifier assignment.

**WT - Connection Weight:** This field is defined as the connection weight. It is used for fair queueing.

**A5 - AAL5 Connection:** This 1-bit field indicates an AAL5 connection if it is 1, and is used by the DQM chip to implement packet-level discarding.

## 4.1.2. External Links

The Dynamic Queue Management Chip has a total output bandwidth of 2.4 Gb/s. The output of the DQM Chip implements four UTOPIA Interfaces (The Universal Test & Operations PHY Interface for ATM), each of which can support a 600 Mb/s OC-12 link or four 150 Mb/s OC-3 links. To support a G-link interface in a nonblocking fashion, a pair of interfaces is allocated, with one actually used for the G-link, and the other disabled. To support an OC-48 link in a nonblocking fashion, two interfaces are used and the other two are disabled.

Figure 4-2 illustrates some of the link configurations that are possible with the DQM chip. Figure 4-2 (a) shows a DQM Chip connecting to 16 OC-3 links. Figure 4-2 (b) shows a DQM Chip connecting to 4 OC-3 links, 1 OC-12 link and 1 G-link. Figure 4-2 (c) shows the connection to a single OC-48 link.

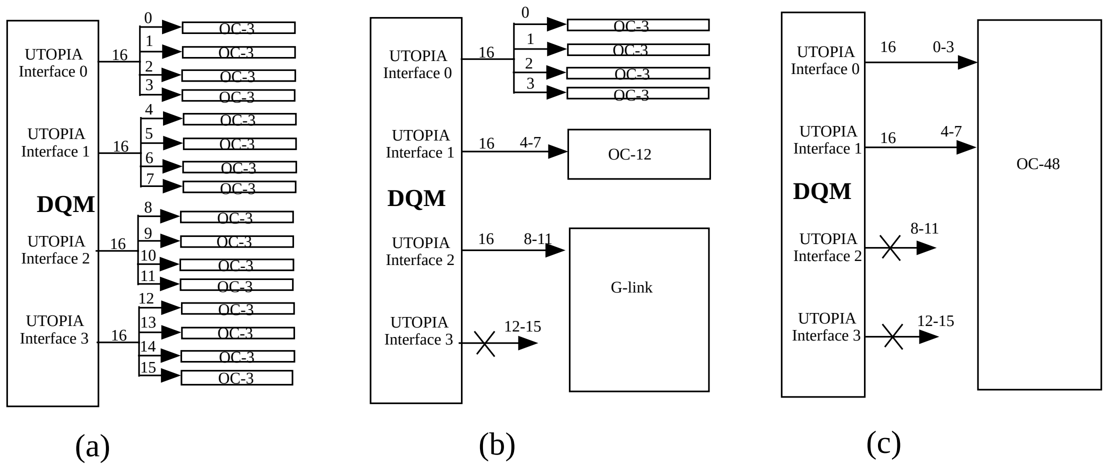


Figure 4-2 Link Configurations

If links are configured in this way, every 16 cell times, OC-3, OC-12, G-link, and OC-48 links can receive one cell, four cells, eight cells and sixteen cells, respectively. We introduce the concept of *Virtual Port*. Each virtual port has a bandwidth of 155Mb/s, which is the same as the bandwidth of an OC-3 link. Figure 4-2 shows the virtual port numbers associated with physical links in each of the sample configurations. The G-link and OC-48 links, in these examples, are associated with the virtual port numbers of the disabled interfaces, as well as the interface they actually use.

If virtual ports are serviced in the normal consecutive order, bursts of cells from single links are created, causing cells to accumulate at link interfaces. However, if we use a reversed-bit counter, the forwarding of cells to links will be spread out in time. Table 4-1 shows the values of the regular counter, the reversed-bit counter and corresponding selected virtual port. The reversed-bit counter can be implemented using a 4-bit regular counter with reversed bit order.

**Table 4-1 Reversed-bit Counter**

| Order | Regular Counter | Reversed-bit Counter | Virtual Port |
|---|---|---|---|
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 1000 | 8 |
| 2 | 0010 | 0100 | 4 |
| 3 | 0011 | 1100 | 12 |
| 4 | 0100 | 0010 | 2 |
| 5 | 0101 | 1010 | 10 |
| 6 | 0110 | 0110 | 6 |
| 7 | 0111 | 1110 | 14 |
| 8 | 1000 | 0001 | 1 |
| 9 | 1001 | 1001 | 9 |
| 10 | 1010 | 0101 | 5 |
| 11 | 1011 | 1101 | 13 |
| 12 | 1100 | 0011 | 3 |
| 13 | 1101 | 1011 | 11 |
| 14 | 1110 | 0111 | 7 |
| 15 | 1111 | 1111 | 15 |

When a virtual port is selected, the actual external link number can be obtained by masking the lower bits of the virtual ports, assuming the external link always connects to the first virtual port. Table 4-2 summarizes the characteristics of different links.

**Table 4-2 Link Characteristics**

| Link Type | OC-3 | OC-12 | G link | OC-48 |
|---|---|---|---|---|
| Bandwidth | 150 Mb/s | 600 Mb/s | 1.2 Gb/s | 2.4 Gb/s |
| Number of Associated Virtual Ports | 1 | 4 | 8 | 16 |
| Mask | 1111 | 1100 | 1000 | 0000 |

# 4.2. System Operation

## 4.2.1. Cell Arrival and Departure

Before we define the major components in the chip, we describe the events involved in processing of arriving and departing cells.

### <u>ARRIVING CELL PROCESSING</u>

**Step One: Decide which queue to append the cell to.** Queues are assigned on a per connection basis. The Queue Selector statically assigns queues to virtual path connections and dynamically assigns queues to virtual circuit connections. When a cell in a virtual circuit connection arrives and finds no matching queue, the Queue Selector assigns an unused queue to the connection. Since queues are distinguished by queue identifiers, determining a queue is the same as producing a queue identifier. The Queue Selector passes the queue identifier to other logic blocks for further processing.

**Step two: Determine the cell slot that the arriving cell should be placed in.** Each arriving cell is temporarily stored in the external memory (Cell Store) before it is sent to the output link. Queues in the Cell Store are implemented as linked lists. The Queue Manager maintains pointers to the first and last slot for each queue. To minimize the required number of external memory operations, an empty slot is included at the end of every list. So, the cell slot to store an arriving cell is the slot at the end of the queue.

**Step three: Add a new cell slot to the end of the queue.** The Queue Manager requests a free slot from the Free Slot Manager. The Free Slot Manager returns the first available slot from the Free Slot List, where all unused slots are stored. The Queue Manager appends that free slot to the end of the queue.

**Step four: Write the arriving cell and the pointer to the empty slot just allocated into the cell slot in the Cell Store.** The Queue Manager sends the slot number and the pointer to the Memory Controller. The Input Master passes the entire cell to the Memory Controller. The Memory Controller generates all the control signals and addresses for a memory write, and writes the cell and the pointer field into the Cell Store.

**Step five: Add the queue to the Output Scheduler if it is a new connection.** The Output Scheduler keeps lists for all active queues. The Queue Manager detects a new connection by comparing the pointer values for the first and the last cells in the queue. The Output Scheduler inserts the new queue into the proper place based on the priority and the output number of the cell.

## DEPARTING CELL PROCESSING

**Step One: Determine the current output and the queue to forward a cell.** The Output Scheduler selects the current output number using a generalized form of round-robin scheduling. It maintains a high priority list and a low priority list for each output. Queues on the lists are served in a round-robin fashion, but the low priority list is served only if the high priority list for the current output is empty.

**Stop two: Determine the cell on the scheduled queue that goes to the link and its position in the Cell Store.** The selected queue always forwards its first cell to the output link. The slot number of the first cell is stored in the FIRST field of the corresponding queue. The Queue Manager uses the queue identifier passed by the Output Scheduler to obtain the slot number.

**Step three: Read the cell out of the Cell Store, place it in the buffer of the Output Master, and return the slot to the free slot cache.** The Queue Manager sends the cell slot number of the outgoing cell to the Memory Controller. The Memory Controller generates the address and the control signals for a memory read. The outgoing cell is read out of the Cell Store and placed in the buffers in the Output Master. The slot number of the departing cell is returned to the free slot cache in the Free Slot Manager for future use.

**Step four: Determine the pointer field of the departing cell and use it to update the FIRST field of the queue list in the Queue Manager.** After the first cell in the queue departs, the second cell becomes the first cell. The pointer to the second cell is read out of the Cell Store with the departing cell. The Cell Store passes the pointer to the Queue Manager to update the FIRST field.

**Step Five: Remove a queue from the Output Scheduler and the Queue Selector if it is empty.** The Queue Manager checks if a queue is empty by comparing the FIRST and the LAST field. For an empty queue, the Queue Selector use the Address Map to find the corresponding entry in the Lookup Table, and sets the valid bit to 0 and the Output Scheduler removes a queue from its list by redirecting the pointers in its queue list.

Table 4-3 and Table 4-4 summarize the events and the components involved in the arriving and departing cell processing. The labels of the events are used later in the timing analysis.

**Table 4-3 Incoming Cell Processing**

| **Labels** | **Descriptions** | QSEL | QMGR | CSTR | OSCHL | FSMGR | FSLST |
|---|---|---|---|---|---|---|---|
| 1 | Determine what queue to append cell to. | + | | | | | |
| 2 | Determine the cell slot. | | + | | | | |
| 3 | If it is a new connection, add the queue to output list. | | + | | + | | |
| 4 | Add a new empty slot to the end of the queue. | | + | | | + | + |
| 5 | Write PTR field and the cell into the cell slot | | | + | | | |

**Table 4-4 Outgoing Cell Processing**

| Labels | Descriptions | QSEL | QMGR | CSTR | OSCHL | FSMGR | FSLST |
|---|---|---|---|---|---|---|---|
| a | Determine the next queue to send from the current output. | | | | + | | |
| b | Determine the cell slot of the next cell to send | | + | | | | |
| c | Read the cell from the memory, place it in the output buffer, and return the slot to the free slot list. | | + | + | | + | + |
| d | Use PTR field of the departing cell to update the FIRST field of the queue | | + | | | | |
| e | If the queue is empty, remove it from the priority list in the Output Scheduler. If it is a VC queue, deallocate the queue and remove the mapping in the Queue Selector. | + | + | | + | | |

## 4.2.2. Data Flows

In this section, we use a data flow diagram to illustrate how components work together. Figure 4-3 shows the data path between the components logically.

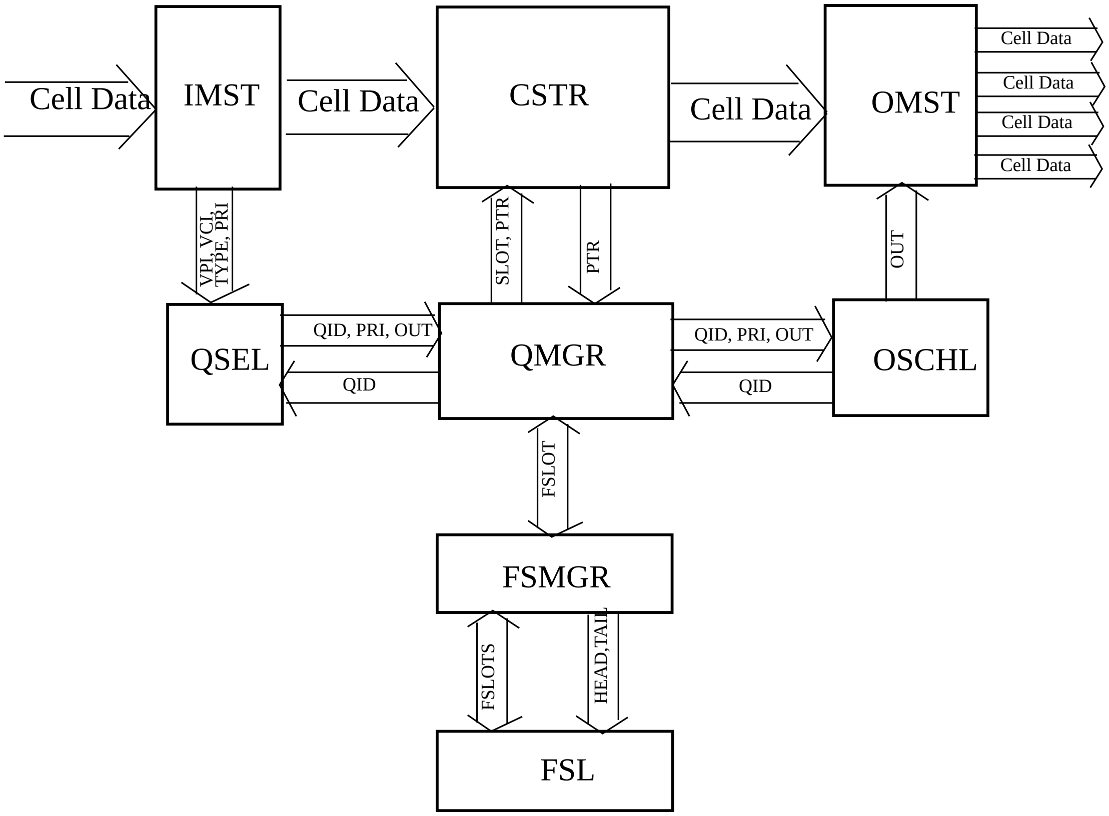


Figure 4-3  Data Flow Diagram

The major data flow is the cell data. Cells received by the IMST are sent to the CSTR first. Cells stored in the CSTR are then sent to the OMST. The OMST forwards cells to different output links.

Queue identifiers (QID) are passed back and forth among the QSEL, the QMGR and the OSCHL for different purposes. To process an arriving cell, the QSEL produces a queue identifier that identifies the queue for this cell. It passes the queue identifier to the QMGR. The QMGR uses this queue identifier to find the cell slot in the CSTR for the incoming cell. If the incoming cell belongs to a new connection, the QMGR also passes the queue identifier to the OSCHL so that the OSCHL can add the queue to its scheduling list. To process a departing cell, the OSCHL informs the QMGR which queue is to forward a cell by passing the corresponding queue identifier to the QMGR. If the queue becomes empty after outputting a cell, the QMGR passes the queue identifier of the empty queue to both the QSEL and the OSCHL. The QSEL and the QSCHL then remove the queue from their lists.

Free slots include all the currently unused cell slots in the CSTR. The FSMGR keeps a local free slot cache. All the remaining free slots are stored in the FSL which is also managed by the FSMGR. The QMGR obtains and returns a free slot through the FSMGR. The head and tail pointers passing from the FSMGR to the FSL indicate the first and the last free slots in the FSL.

The QMGR informs the CSTR where to store a cell or read a cell from by sending the slot number (SLOT). The pointer field (PTR) passed by the QMGR is stored in the cell slot with the cell to build up the linked list. When the cell departs, the pointer field of the cell slot is passed back to the QMGR to update the queue list.

## 4.3. Component Definitions

To define the signals between the components, we use the convention that a signal consists two parts separated by an underscore. The name before the underscore indicates the signal, and the component name after the underscore indicates where the signal is generated. For example, VPI_IMST means the VPI signal generated in the IMST.

Let NVCI be the number of VC queues supported by the DQM chip. Signal definitions of all the components are in Appendix A.

### 4.3.1. Input Master (IMST)

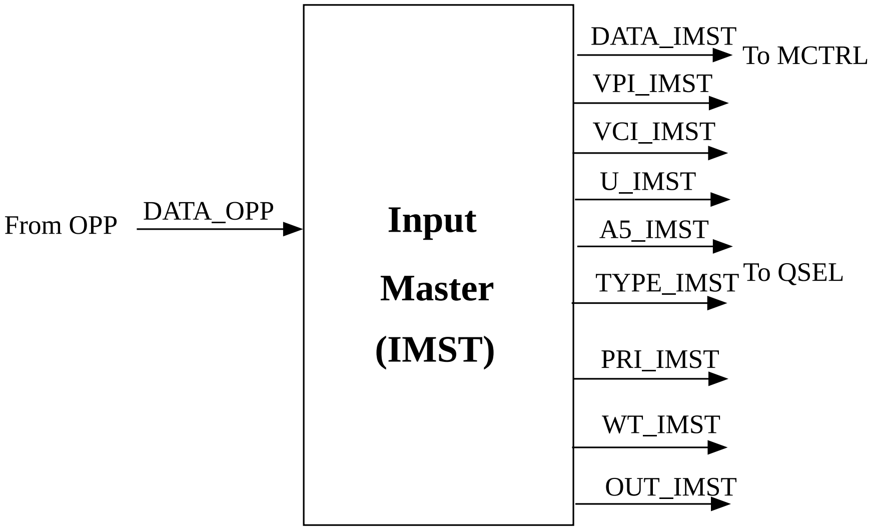


Figure 4-4 Signal Diagram for the Input Master

The signal diagram for the Input Master (IMST) is shown in Figure 4-4. The IMST receives cells from the OPP in a format shown in Figure 3-1 on page 19. It passes cell information to the Queue Selector (QSEL) and passes the entire cell unchanged to the Memory Controller (MCTRL).

Figure 4-5 shows the logic in the IMST. The IMST extracts VPI_IMST, VCI_IMST, A5_IMST, U_IMST, TYPE_IMST, PRI_IMST, WT_IMST and OUT_IMST from the corresponding fields in the cell and sends them to the QSEL. It copies DATA_OPP to DATA_IMST and forwards it to the MCTRL.

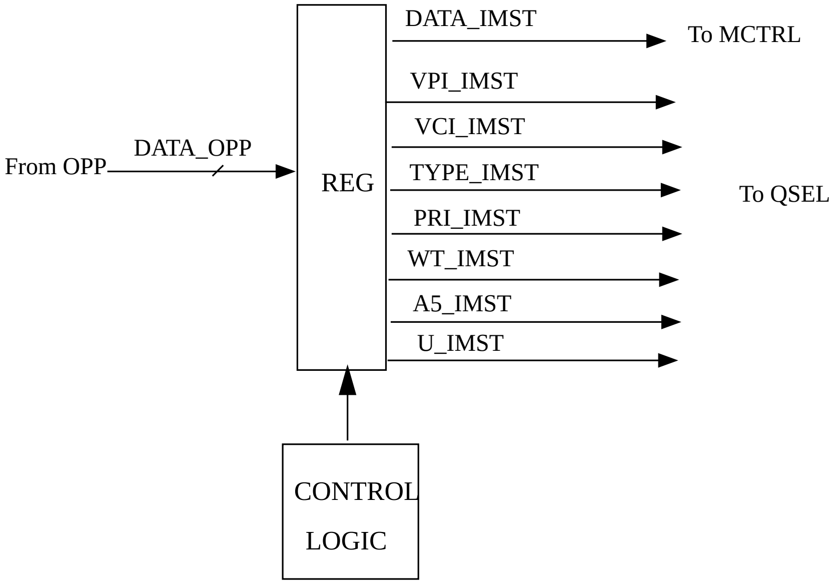


Figure 4-5  Block Diagram of the Input Master

## 4.3.2. Queue Selector (QSEL)

Figure 4-6 shows the signal diagram of the Queue Selector (QSEL). The QSEL receives cell information from the Input Master (IMST) and assigns a queue identifier to the cell. It sends the queue identifier and some cell information to the Queue Manager (QMGR). The QSEL receives control signals from the QMGR and removes queue mapping as needed.

Figure 4-7 illustrates the internal logic structures of the QSEL. The QSEL keeps a set-associative memory (SAM) and a content addressable memory (CAM) for dynamic queue assignment. The Free VC Queue List contains all unused queue identifiers. The Address Map is used to remove empty queues from corresponding entries.

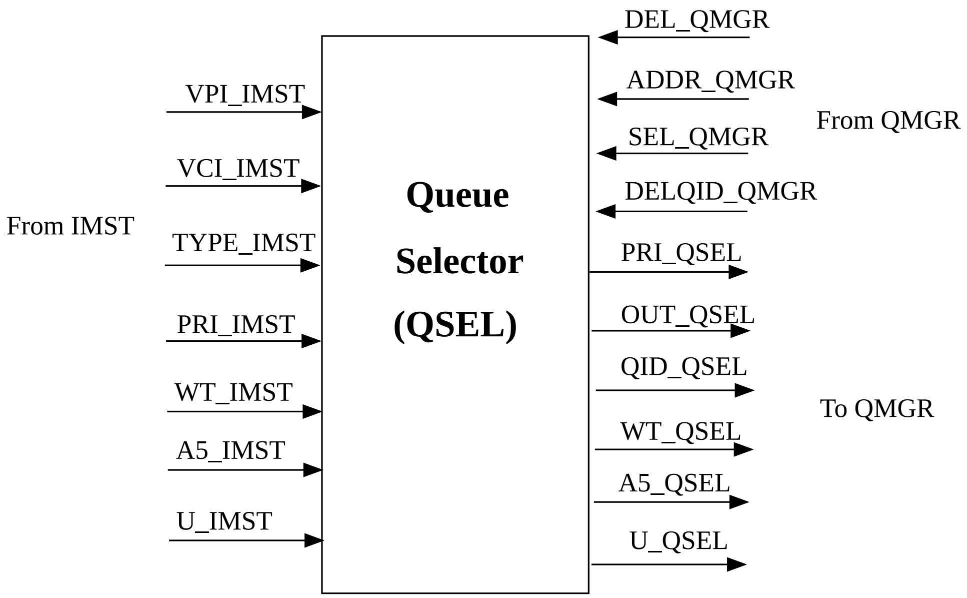


Figure 4-6  Signal Diagram for the Queue Selector

If TYPE_IMST is 0, the incoming cell is from a VP connection. A queue identifier of value (VPI + NVCI) is statically assigned to the cell.

If TYPE_IMST is 1, the incoming cell belongs to a VC connection. Dynamic queue assignment is used in this case. Dynamic queue assignment is implemented using a SAM and a CAM. Each entry of a SAM contains a valid bit, a tag field and a queue identifier. The total number of entries in the SAM is determined by the maximum number virtual circuits supported by the DQM chip and the load factor. Depending on the depth and the width of the set-associative memory, the address to the SAM (ADDR) is formed from the lower order bits of the VPI and the VCI. The tag of the cell (TAG) is generated using the rest of the bits. The CAM contains a few entries, each of which includes a key field, a queue identifier (QID) field, and a valid bit. The key field stores the VPI and the VCI. The QID field contains the assigned queue identifier if the valid bit is 1. The choices of the maximum number of VCs, the load factor, the width of the SAM and the size of the CAM are discussed in Chapter 6.

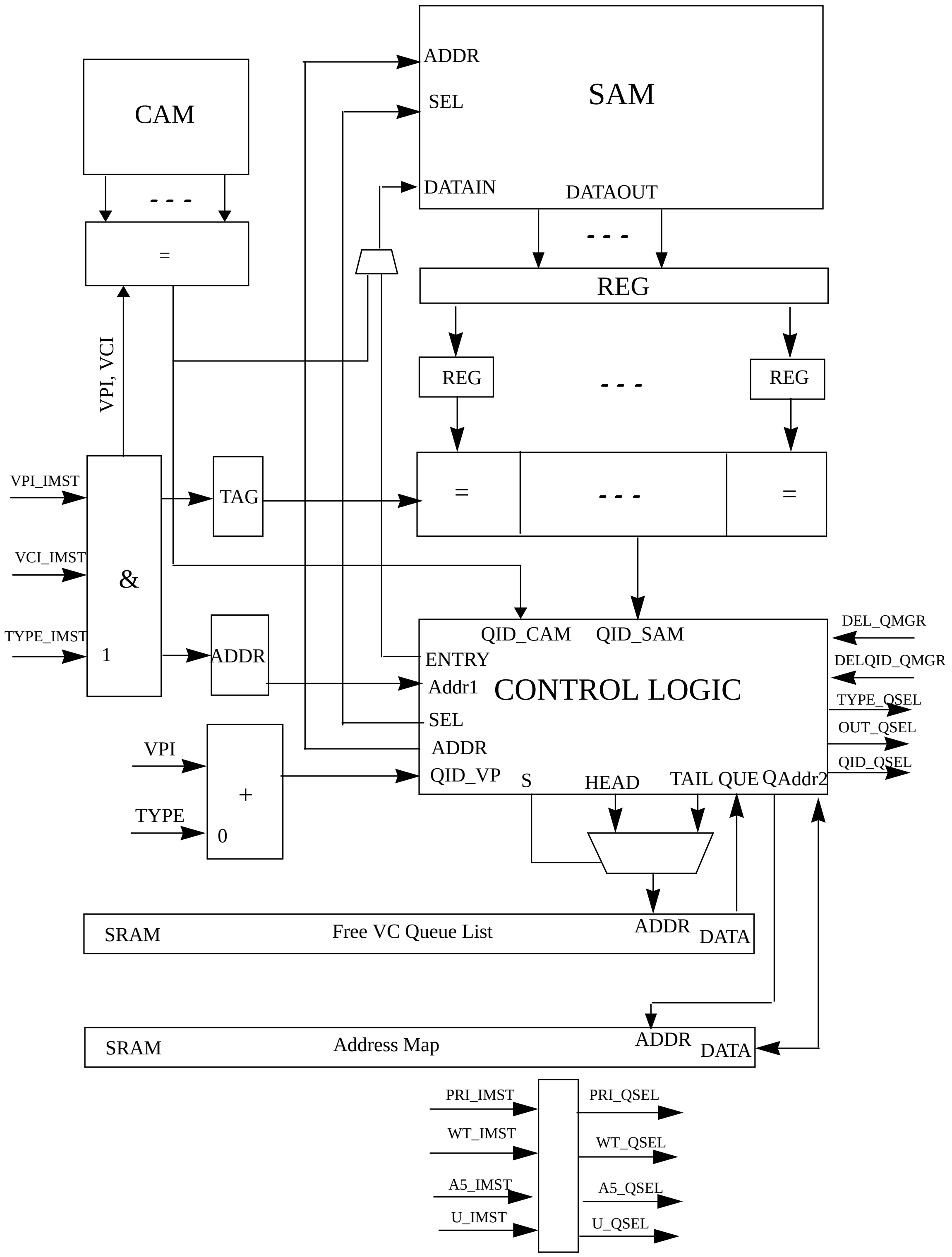


Figure 4-7  Block Diagram of the Queue Selector

ADDR is used to retrieve a set of entries from the SAM. The tag fields in all entries in the selected set are compared to TAG of the incoming cell in parallel. If there is a matched entry, the queue identifier stored in the entry is assigned to the incoming cell.

When the SAM is doing a lookup, the entire VPI and VCI are compared to the key field of all entries in the CAM in parallel. If there is a matching entry in the SAM, the queue identifier is assigned as described above.

If the SAM contains no matching entry, but the CAM has a matching entry, then the queue identifier stored in the CAM entry is assigned to the cell. If the corresponding set in the SAM has an unused entry in this case, the queue identifier in the CAM entry is copied to the SAM entry and the mapping in the CAM is removed by setting the valid bit to 0. In order to remove the mapping in the future, the position of the entry in the SAM (specified by the address to the set and the entry number) is stored in the address map indexed by the queue identifier.

In all other cases, the first unused queue identifier in the free VC queue list is assigned to the cell. If the selected set in the SAM has one or more unused entries, the TAG of the cell and the newly assigned queue identifier are written into the first unoccupied entry in the set. The valid bit is then set to 1. The address map is modified in the same way as previously described. Otherwise, an unused entry in the CAM is allocated. The VPI and the VCI are written into the key field and the newly assigned queue identifier is written into the QID field. The entry location in the CAM is written into the address map.

In all cases, the queue identifier assigned to the incoming cell is sent to the QMGR as QID_QSEL.

The free VC queue list is implemented as a circular list. All unused queue identifiers are stored in the list. The size of the free queue list is equal to the maximum number of VC queues supported by the DQM chip. Each entry in the free queue list contains an unused queue identifier. The control logic keeps track of the head and the tail of the list. HEAD points to the first available queue identifier in the list. To simplify the design, TAIL always points to the first empty entry. After a free queue identifier is read out of the free queue list, the HEAD pointer is incremented by 1 and will wrap around when it reaches the physical end of the memory. A queue identifier is returned to the entry specified by the TAIL pointer. The TAIL pointer is incremented by 1 in this case and will also wrap around as needed.

When a VC queue becomes empty, the queue identifier of the empty queue needs to be returned to the Free VC queue list. The QMGR detects an empty VC queue and provides the queue identifier to the QSEL. The queue identifier is returned to the free VC queue list. In order to remove the mapping in the SAM or the CAM, the entry in the address map is used to set the valid bit to 0 in the corresponding entry.

### 4.3.3. Queue Manager (QMGR)

Figure 4-8 shows the signal diagram of the Queue Manager (QMGR). The QMGR maintains a Queue List that keeps track of all queues in the Cell Store. The queue list is indexed by queue identifiers. Each entry of the queue list contains pointers to the cell slots where the first cell and the last cell of the queue are stored in the Cell Store. The QMGR specifies the cell slot that an incoming cell is written to and the cell slot that a departing cell is read from. The block diagram of the QMGR is illustrated in Figure 4-9.

The queue list is stored in the SRAM. Each entry has a FIRST field and a LAST field. The FIRST field contains the cell slot for the first cell in the queue. The LAST field contains the cell slot that the last cell in the queue points to.

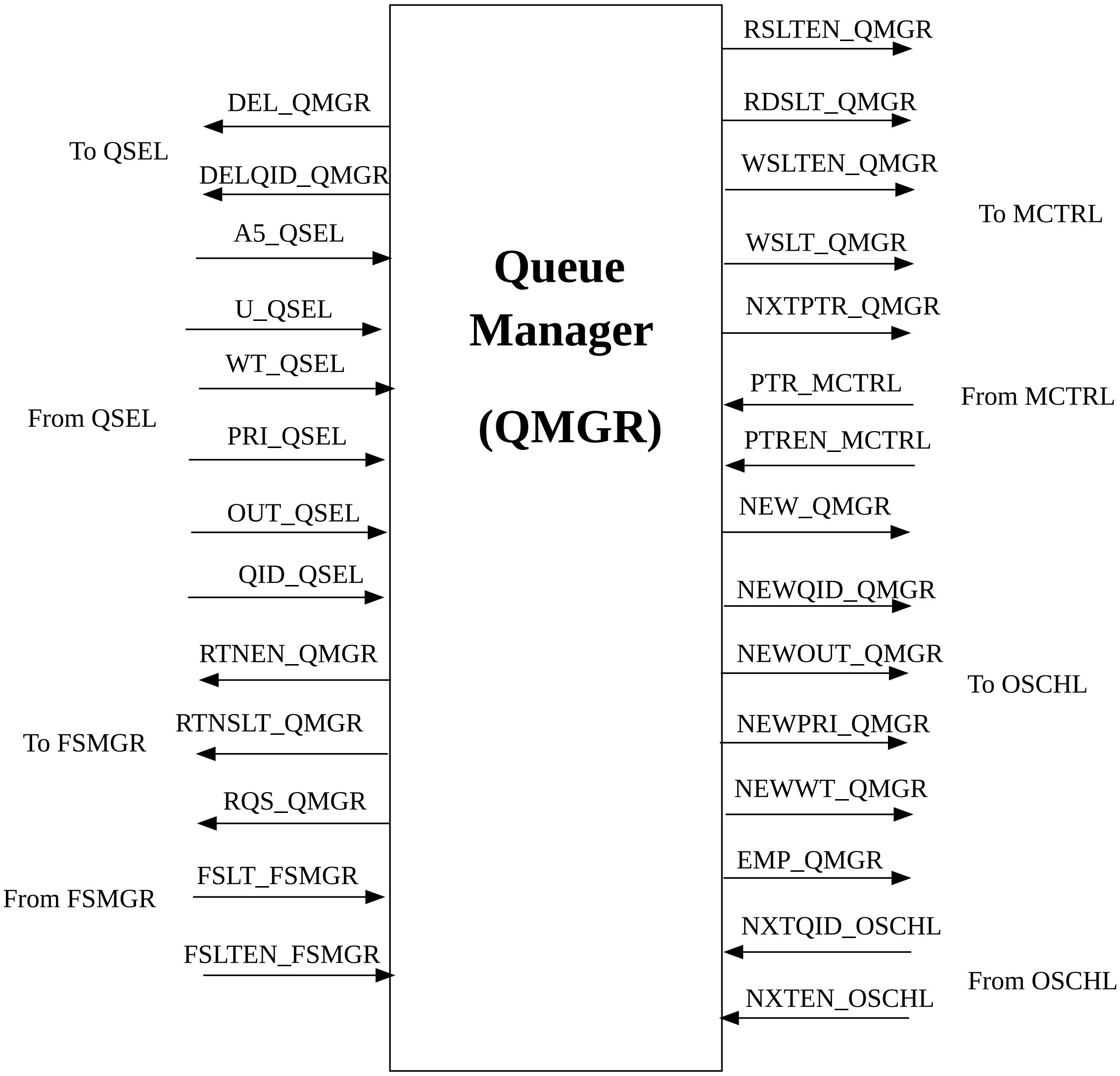


Figure 4-8 Signal Diagram for the Queue Manager

The queue identifier of a incoming cell is passed by the QSEL. The QMGR uses the queue identifier to retrieve the entry in the queue list. The FIRST field and the LAST field are compared. If these two fields are equal, the incoming cell belongs to a new connection. The new queue needs to be added to the priority list in the Output Scheduler (OSCHL). The QMGR sends the queue identifier, the output number, the priority and the weight to the OSCHL.

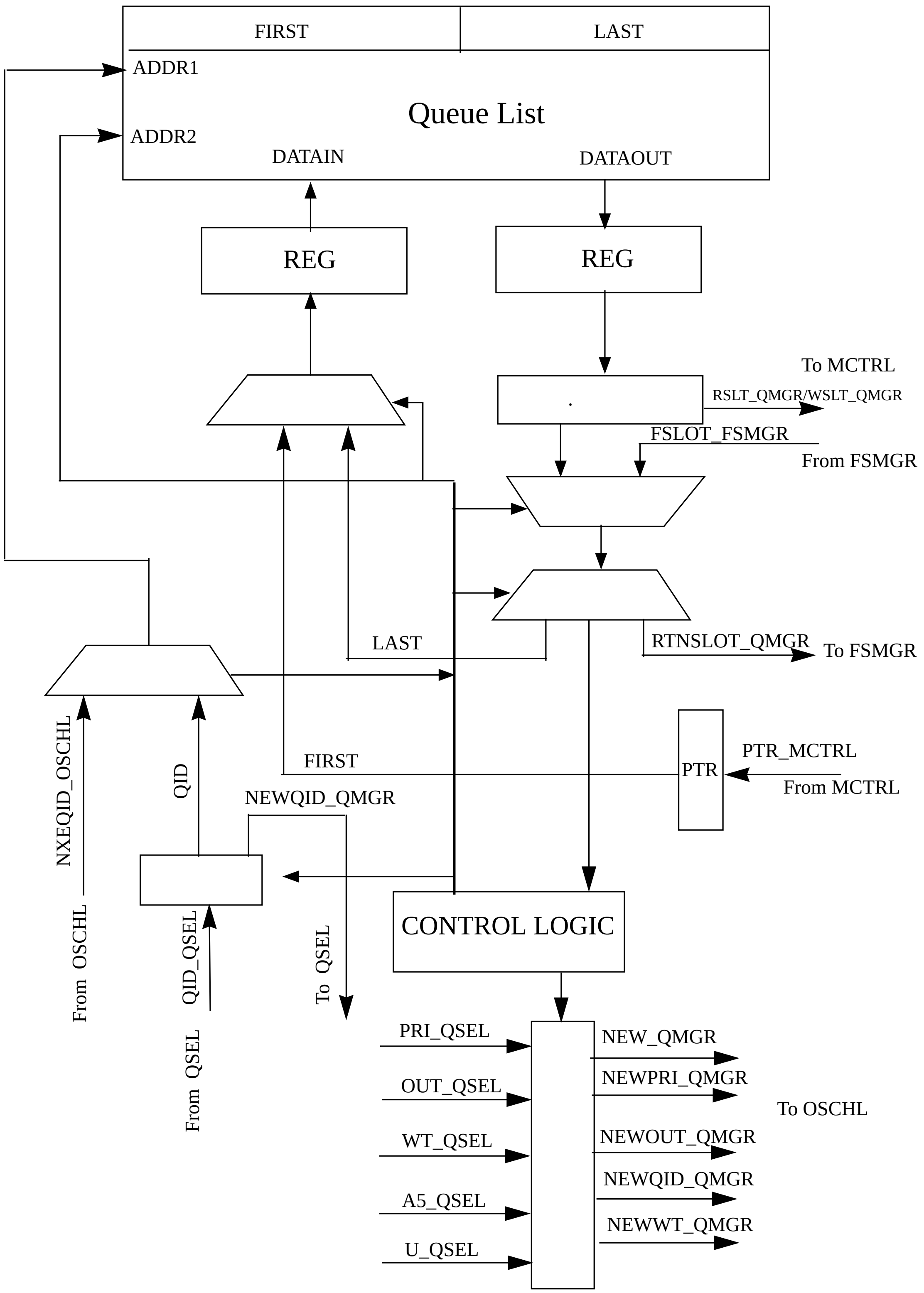


Figure 4-9 Block Diagram of the Queue Manager

The LAST field specifies the cell slot to store the incoming cell. In order to build up the linked list, the QMGR sends a free slot along with the cell slot to the Memory Controller. The free slot is written into the Cell Store with the incoming cell. The QMGR requests another free slot from the Free Slot Manager (FSMGR).

The queue identifier of a departing queue is sent by the Output Scheduler (OSCHL). The QMGR retrieves the corresponding entry in the queue list. The FIRST field contains the cell slot where the first cell in the queue is stored. The first cell of the queue is read out from the Cell Store and is sent to the Output Master. The pointer stored with the cell is sent back to the QMGR to update the FIRST field of the queue list. This pointer is also compared to the LAST field in the entry. If these two values are equal, the queue is empty and needs to be removed. The QMGR then informs the QSEL and the OSCHL to remove the queue from their lists. The cell slot that contains the departing cell becomes free slot and is sent to the FSMGR.

### 4.3.4. Output Scheduler (OSCHL)

Figure 4-10 shows the signal diagram of the Output Scheduler (OSCHL). The OSCHL determines the order of queues to forward cells to output links. Once a queue is selected, it can forward one cell to the output. The basic design describes the two priority scheduling.

The OSCHL supports up to 16 output links. It keeps a high priority list and a low priority list for every output. Priority lists are constructed as circular lists. Once an output is selected, the high priority list is served before the low priority list. Queues on the priority list are served in round-robin fashion.

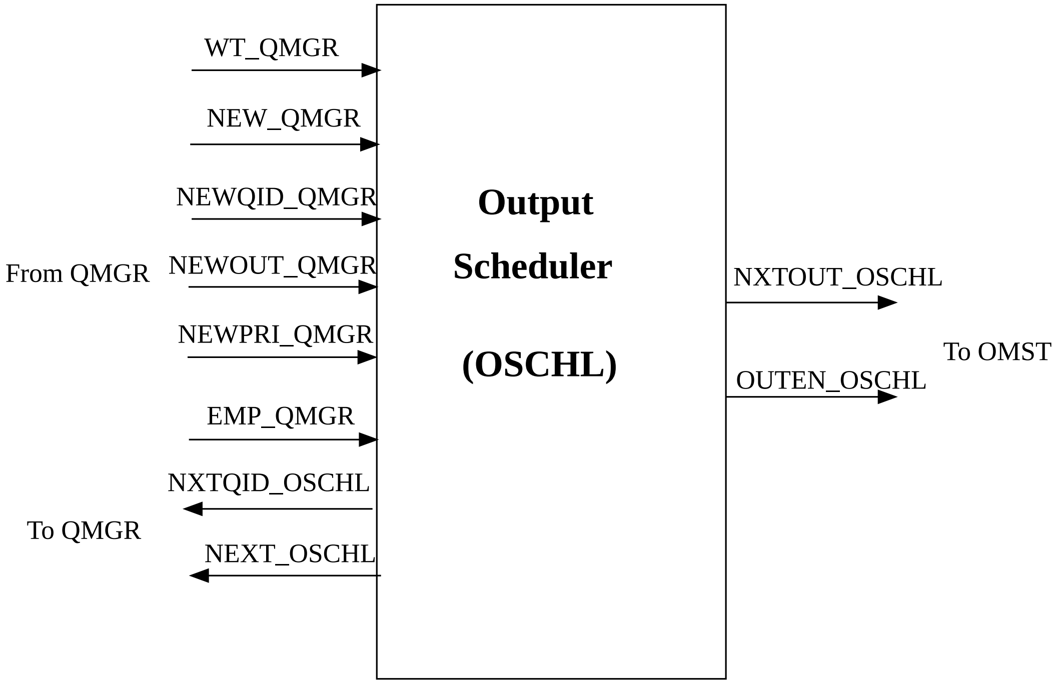


Figure 4-10 Signal Diagram for the Output Scheduler

Figure 4-11 shows the block diagram of the OSCHL. A queue list is stored in SRAM to build the priority lists. The number of entries in the queue list is equal to the total number of connections supported by the DQM chip including the VP connections. The queue list is indexed by the queue identifiers. The value in the entry specifies the next queue in the circular list. Only active queues with at least one cell are on the circular lists. The OSCHL also keeps a scheduling list stored in SRAM with 16 entries. Each entry contains a HIGH pointer to the next queue on the high priority list and a LOW pointer to the next queue on the low priority list. If there is no queue on the list, the pointer is set to NIL.

The OSCHL uses a 4 bit reversed-bit counter to determine the order in which the output links are selected as described in 4.1.2. Once an output is selected, the OSCHL checks the values in the HIGH field and LOW field in the corresponding entry. If HIGH is not NIL, the value in the corresponding queue list entry specifies the next queue to forward a cell. The OSCHL sends the queue identifier of the scheduled queue to the QMGR. If

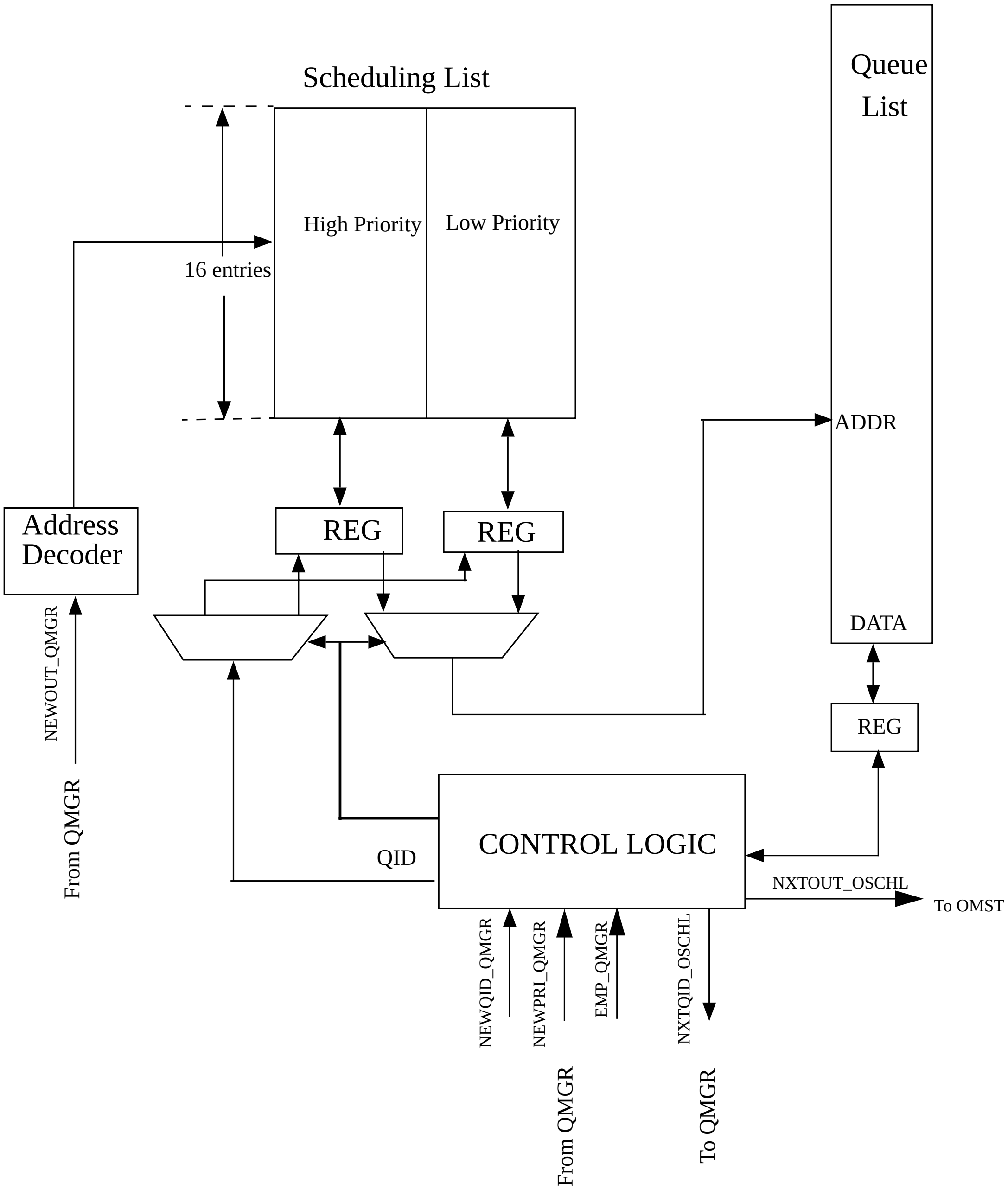


Figure 4-11 Block Diagram of the Output Scheduler

HIGH is NIL, but LOW is not NIL, the OSCHL sends the queue identifier indexed by the LOW field to the QMGR as the next scheduled queue. If LOW is also NIL, no queues are destined for the selected output. The cycle becomes idle and no cell is forwarded to the output.

The OSCHL is responsible for adding new queues to its priority lists and removing empty queues from its priority lists. To add a new queue to the list, the OSCHL uses the output number and the priority of the new queue sent by the QMGR to find the proper list to insert the new queue. The value in the queue list entry indicated by the HIGH/ LOW pointer is copied to the queue list entry indexed by the new queue. The queue identifier of the new queue is then copied to the queue list entry indexed by the HIGH/ LOW field.

The QMGR determines if a queue becomes empty after forwarding a cell. The OSCHL needs to remove the empty queue from its list. The OSCHL copies the value in the queue list entry of the scheduled queue to the entry indexed by the HIGH/LOW field. In this case, the HIGH/LOW field does not change. Otherwise, the value in the queue list entry of the scheduled queue is copied to the HIGH/LOW field. As a result, the pointer points to the next queue in the circular list.

### 4.3.5. Free Slot Manager (FSMGR)

Figure 4-12 is a signal diagram of the Free Slot Manager (FSMGR). The FSMGR keeps an internal recycling cache and manages the Free Slot List (FSL) in the external memory. Figure 4-13 is a block diagram of the FSMGR and its relationship with the FSL. The FSL stores unused cell slots in the Cell Store and is organized as a circular list.

When the Queue Manager (QMGR) requests for a free slot, the FSMGR sends a free slot from its free slot recycling cache. When the QMGR returns a free slot, the FSMGR places it in its recycling cache. In normal cases, where the input rate matches the output rate, there is no need to access off-chip Free Slot List. The FSL only needs to be

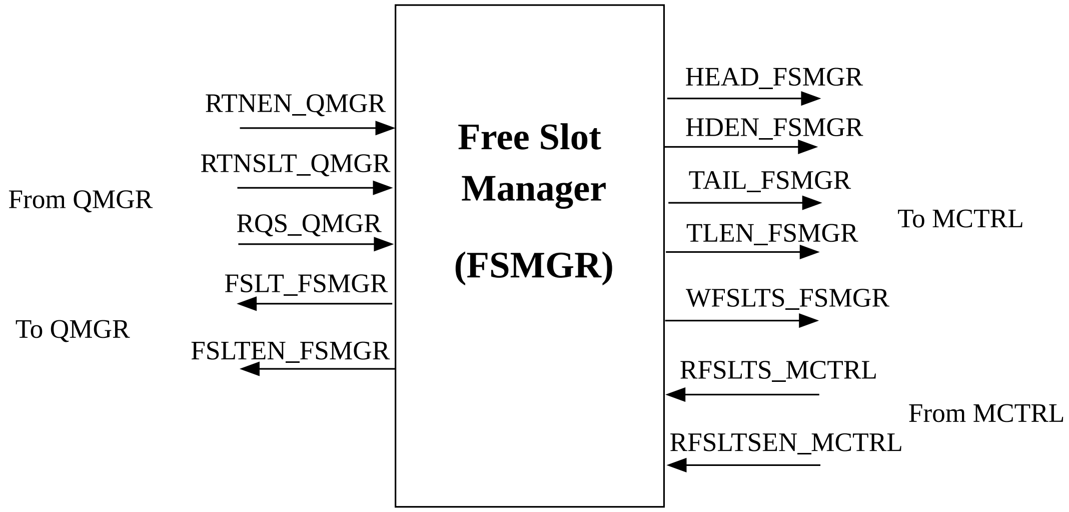


Figure 4-12 Signal Diagram for the Free Slot Manager

accessed to read a block of free slots or to return a block of free slots when there is a mismatch in rates for quite a long time. In this case, there is idle memory cycles available.

### 4.3.6. Memory Controller (MCTRL)

The signal diagram for the Memory Controller (MCTRL) is shown in Figure 4-14. The MCTRL controls the external memory, generates all control signals and does the proper format conversion. Figure 4-15 shows the logic diagram of the Memory Controller. The MCTRL generates all the control signals necessary for the external memory, and does the proper format conversion.

The MCTRL receives the incoming cell from the Input Master (IMST) on a 32 bit data path, converts it to a 160 bit data path. It receives the cell slot to store the cell sent by the Queue Manager (QMGR), and translates it to the physical memory address. The incoming cell is written to the cell slot in the Cell Store (CSTR). The MCTRL also

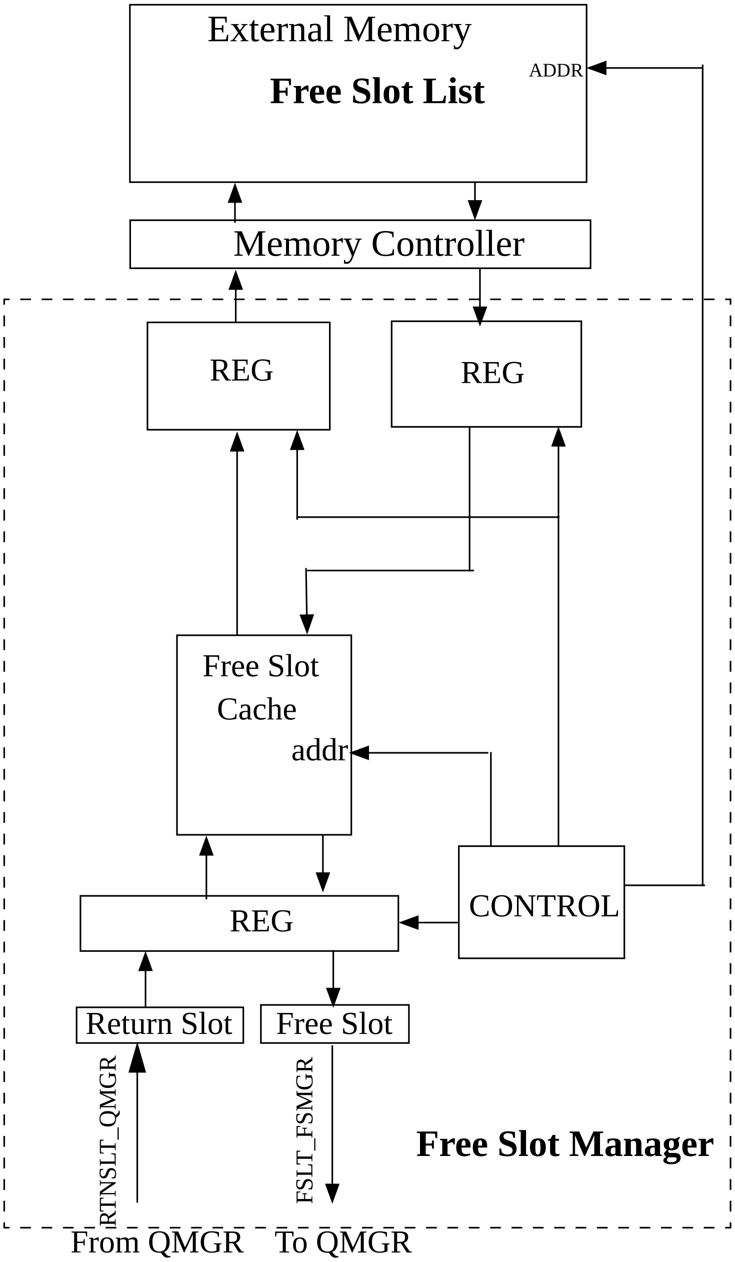


Figure 4-13 Block Diagram of the Free Slot Manager

receives the cell slot for a departing cell from the QMGR, does the address translation and reads the cell from the CSTR. The MCTRL sends the pointer field to the QMGR and the cell to the Output Master (OMST) on a 32 bit data path after format conversion. The MCTRL receives free slots from the FSMGR, and writes them to the Free Slot List

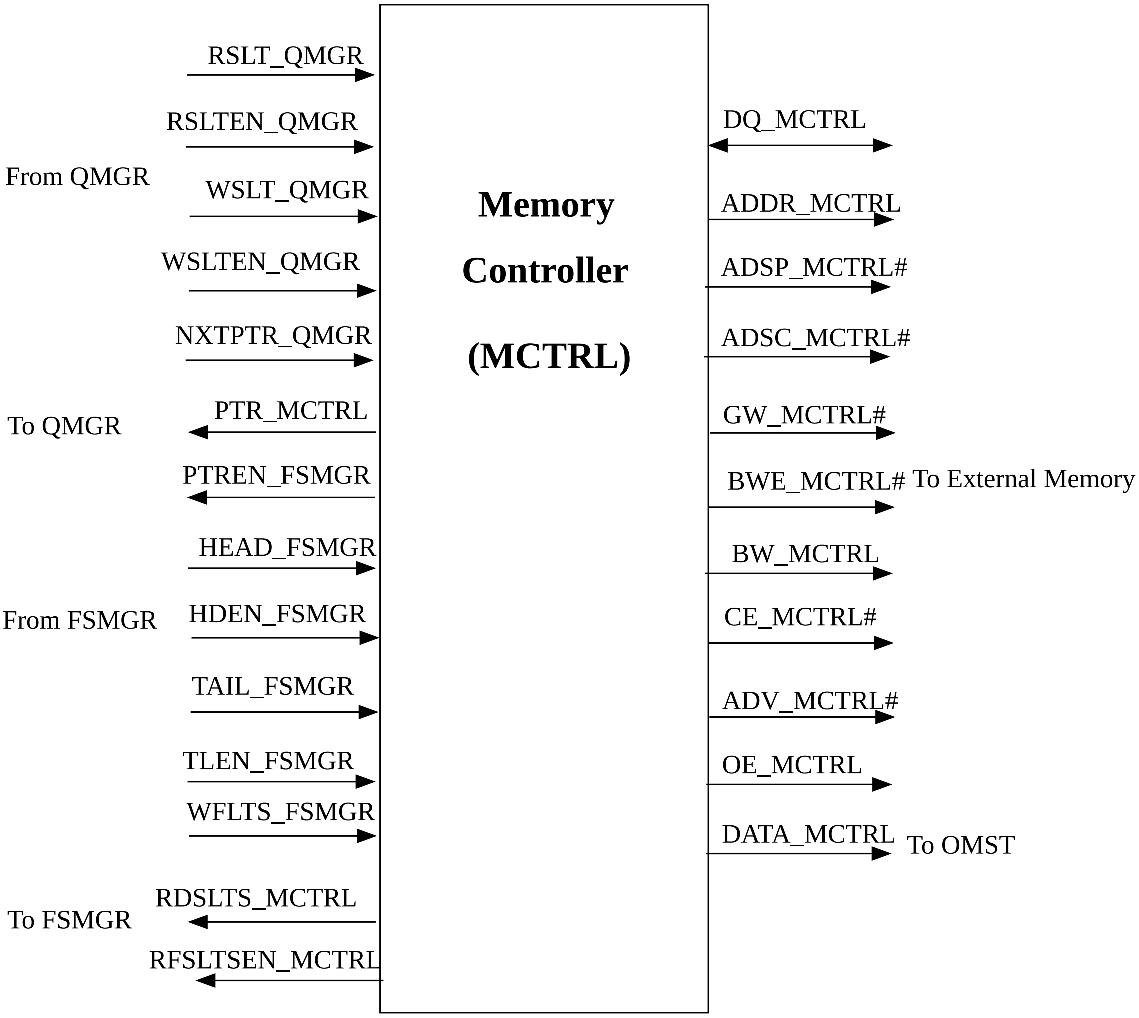


Figure 4-14 Signal Diagram for the Memory Controller

(FSL). When free slots are read out from the FSL, the MCTRL converts the format and sends to the FSMGR.

### 4.3.7. Output Master (OMST)

Figure 4-16 shows the signal diagram of the Output Master (OMST). The OMST receives outgoing cells from the Memory Controller, and buffers they before being forwarded to external links. The logic diagram is shown in Figure 4-17. It implements

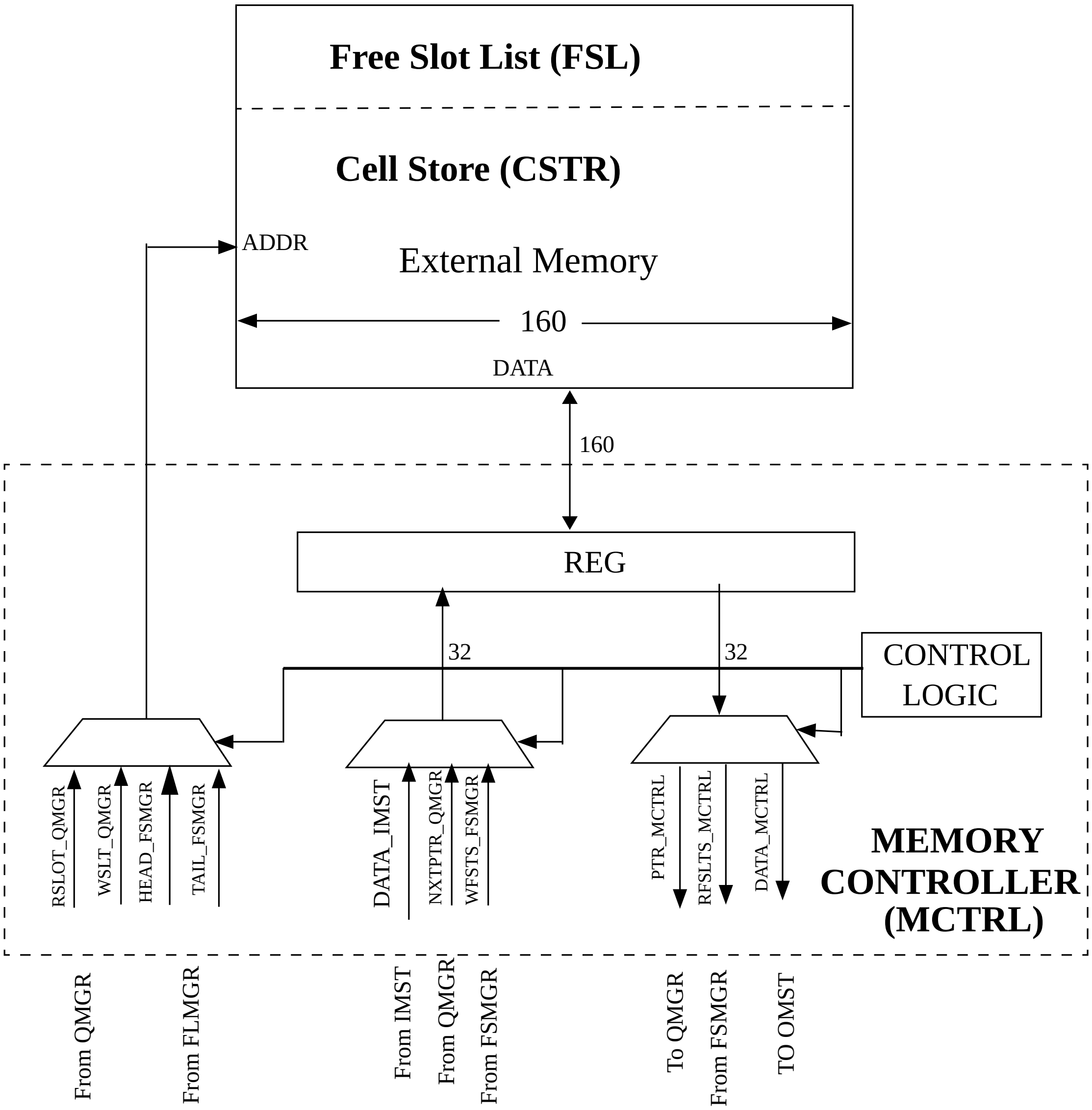


Figure 4-15 Block Diagram of the Memory Controller

four UTOPIA interfaces. The OMST has four FIFOs in SRAM, one for each UTOPIA interface. Each FIFO can hold eight cells.

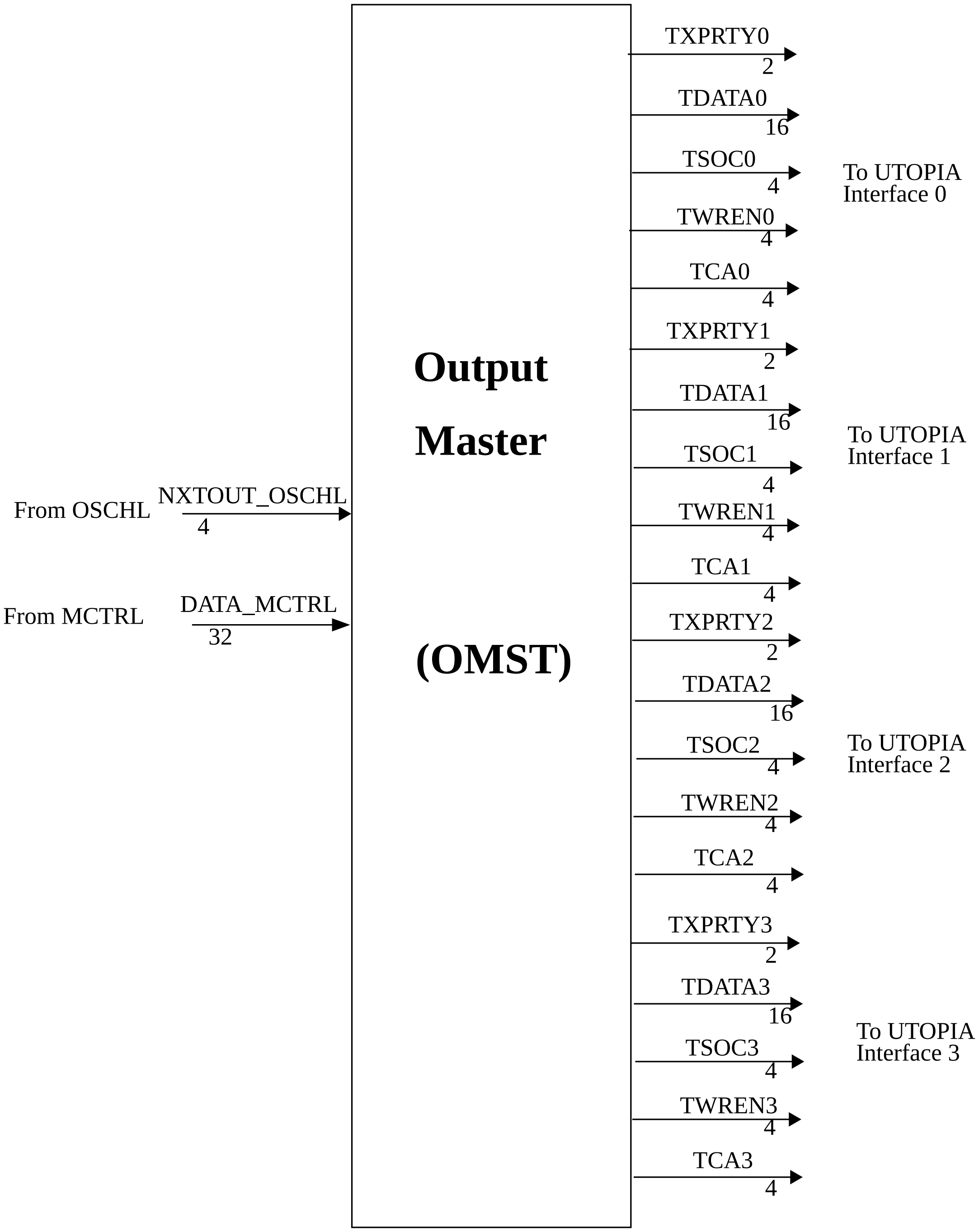


Figure 4-16  Signal Diagram for the Output Master

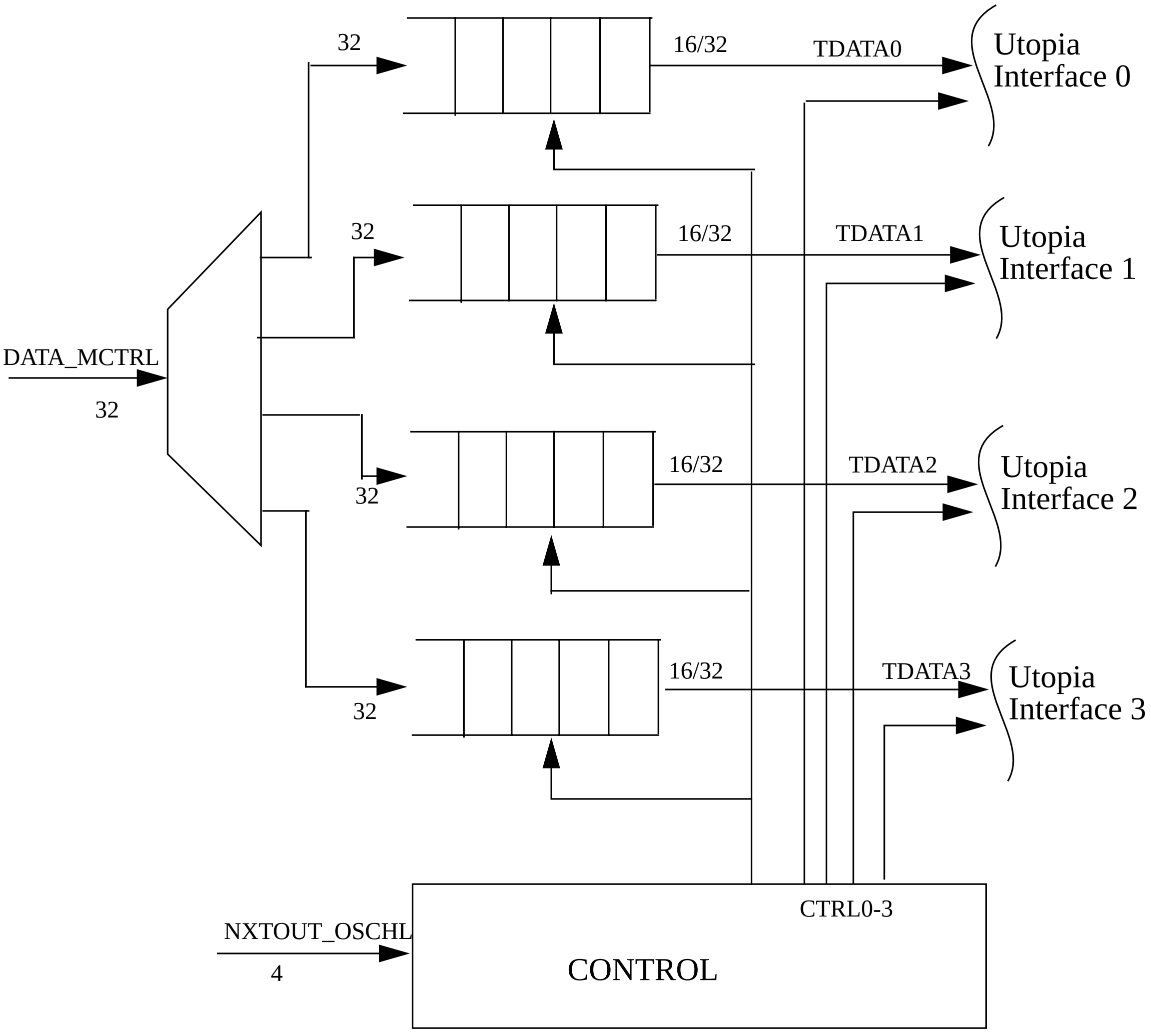


Figure 4-17 Block Diagram of Output Master

## 4.4. Timing Analysis

In this section, we show that all operations can meet timing requirements. Since each event involves operations of several blocks, these events need to be subdivided into mini-events in order to develop detailed timing diagrams. Table 4-5 and Table 4-6 list all the major events and mini-events involved in processing arriving and departing cells.

**Table 4-5 Detailed Incoming Cell Processing**

| **Symbol** | **Descriptions** | QSEL | QMGR | CSTR | OSCHL | FSMGR |
|---|---|---|---|---|---|---|
| 1 | Determine what queue to append cell. | + | | | | |
| 2 | Determine the cell slot. | | + | | | |
| 3 | If it is a new connection, add the queue to output list. | | + | | + | |
| 3.1 | Detect a new connection | | + | | | |
| 3.2 | Add queue to the priority list. | | | | + | |
| 4 | Add a new empty slot to the end of the queue. | | + | | | + |
| 4.1 | Send an empty slot as PTR to CSTR. | | + | | | |
| 4.2 | Update Last pointer | | + | | | |
| 4.3 | Send a free slot to QMGR | | | | | + |
| 5 | Write PTR field and the cell into the cell slot. | | | + | | |

Some of the events depend on the completion of other events. In order to make the sequencing correct, we use directed graphs to show the dependency among events. Figure 4-18 and Figure 4-19 are dependency graphs for the incoming and outgoing cell processing.

**Table 4-6 Detailed Outgoing Cell Processing**

| Symbol | Descriptions | QSEL | QMGR | CSTR | OSCHL | FSMGR |
|---|---|---|---|---|---|---|
| a | Determine the next queue to send from the current output. | | | | + | |
| b | Determine the cell slot of the next cell to send | | + | | | |
| c | Read the cell from the memory, place it in the output buffer, and return the slot to the free slot list. | | | + | | + |
| c.1 | Read PTR field from memory | | | + | | |
| c.2 | Read cell from memory | | | + | | |
| c.3 | Return the slot of the outgoing cell. | | + | | | |
| c.4 | Store the returned slot. | | | | | + |
| d | Use PTR field of the departing cell to update the FIRST field of the queue. | | + | | | |
| e | If the queue is empty, remove it from the priority list in the Output Scheduler. If it is a VC queue, deallocate the queue and remove the mapping in the Queue Selector. | + | + | | + | |
| e.1 | Detect an empty queue. | | + | | | |
| e.2 | Return the queue to the free queue list, remove the mapping. | + | | | | |
| e.3 | Remove the queue from the output list. | | | | + | |

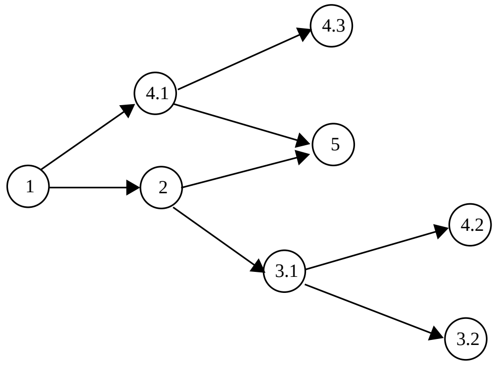


Figure 4-18 Detailed Dependency Graph for Incoming Cell Processing

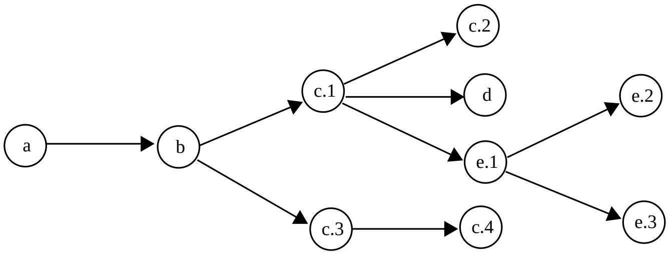


Figure 4-19 Detailed Dependency Graph for Outgoing Cell Processing

Another constraint that makes timing tight is the on chip memory accesses. Within each block, memory accesses need to be done sequentially. Table 4-7 shows the events that involve memory accesses. Figure 4-20 is the detailed timing diagram of the chip.

**Table 4-7 Events Involving Memory Accesses**

| | |
|---|---|
| Queue Manager | 2; 4.2; b; d |
| Queue Selector | 1; e |
| Output Scheduler | 3.2; a; e |
| Free Slot Manager | 4.3; c.4 |

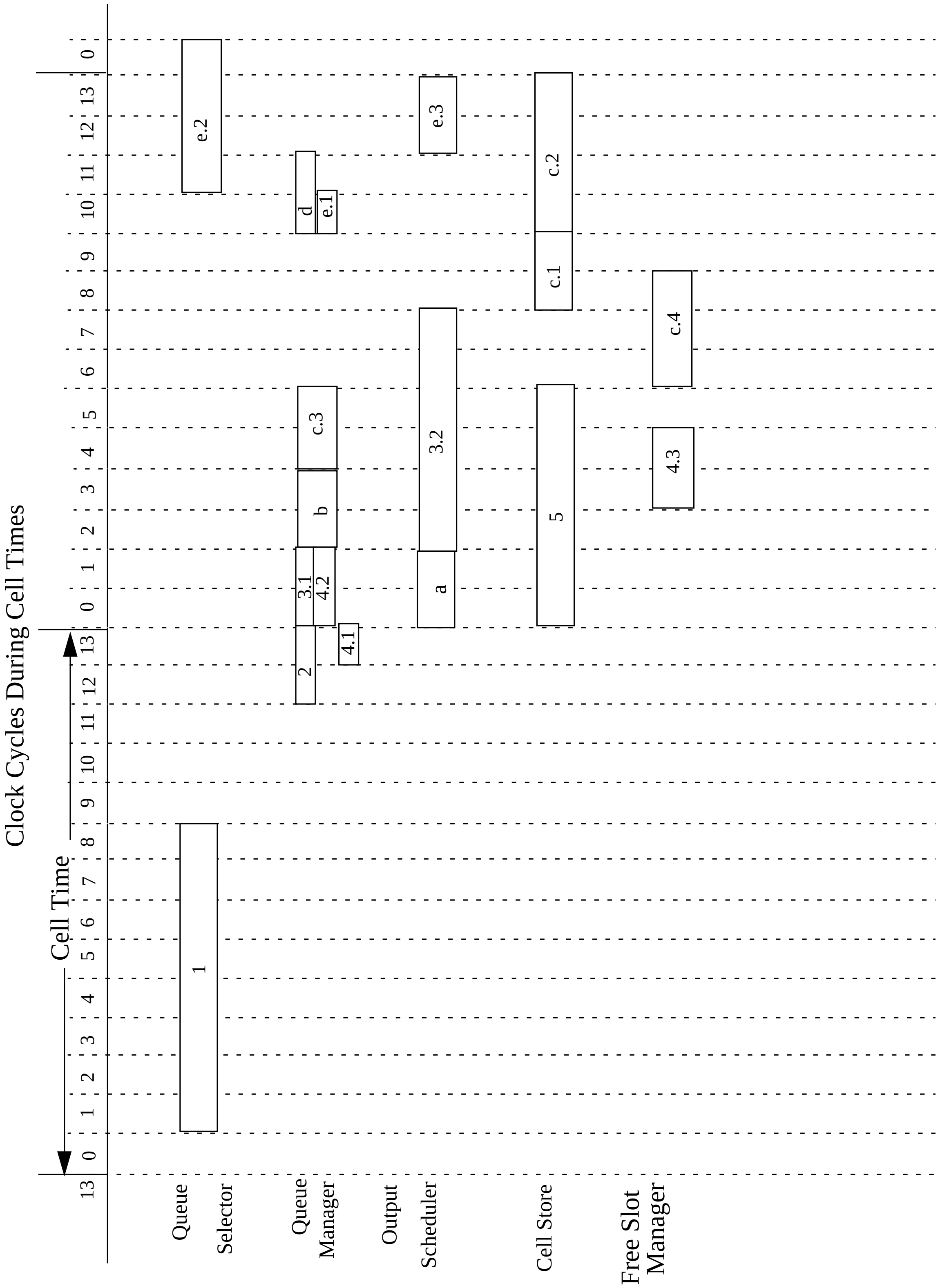


Figure 4-20  Detailed Timing Diagram

# 5. ADVANCED ALGORITHMS

## 5.1. Binary Scheduling Wheels Algorithm

Two priority levels can be used to distinguish real-time traffic from non-real-time traffic, and minimize delay for real-time traffic. However, we may require greater flexibility in allocating bandwidth among virtual circuits. In this case, weighted round-robin scheduling can be used to allocate bandwidth among virtual circuits. The *Binary Scheduling Wheels* (BSW) algorithm used in the DQM implements weighted round-robin scheduling at minimal cost, providing a wide range of rate options. In addition, because bursty virtual circuits with high peak-to-average ratio are more likely to cause congestion in the downstream switches, the BSW algorithm distributes cells from the same channel evenly, minimizing the burstiness of the output streams.

### 5.1.1. Binary Scheduling Wheels

The Output Scheduler uses the per VC based Binary Scheduling Wheels algorithm to implement weighted round-robin scheduling in a very cost efficient way. All virtual circuits have power of 2 weights and during overload periods, share the link bandwidth in proportion to their weights. Instead of forwarding as many cells as specified by the weight once a queue is selected, the BSW algorithm places queues on scheduling wheels with different weights and alternates among wheels.

For implementation efficiency, we restrict weights to be powers of 2. Suppose we support $W$ different weights: $2^0, 2^1, 2^2, \ldots, 2^{W-1}$. We construct $W$ binary scheduling wheels, one for each weight factor. Each VC queue is placed on a corresponding scheduling wheel. The scheduling wheel with weight $2^0$ is visited twice as frequently as the scheduling wheel with weight $2^1$, four times as frequently as the one with weight $2^2$, and so forth. $W$ power of 2 weights can be coded using $\log W$ bits.

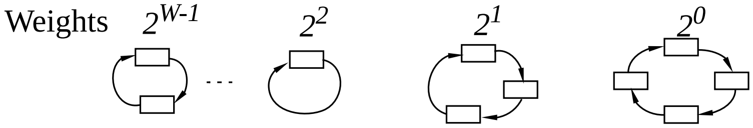


Figure 5-1 Binary Scheduling Wheels

Figure 5-1 shows an example with $W$ binary scheduling wheels. Each little box in the figure represents a list node containing a queue identifier that identifies a non-empty per virtual circuit queue. Once a scheduling wheel is selected, all queues on that scheduling wheel can forward one cell to the output.

A $W$-bit binary counter can be used to select binary scheduling wheels with $W$ weights. In a $W$-bit binary counter, the least significant bit of a binary counter changes twice as fast as the next lowest order bit, four times as fast as the next bit, and so forth. This property matches nicely with our scheduling wheel selection procedure.

Suppose the counter value is $C = c_{W-1} \ldots c_1 c_0$. When the counter advances, a change in bit $i$ triggers servicing of the scheduling wheel with weight $2^i$. We organize the transmission schedule into a series of passes. At the start of each pass, the counter is incremented. During the pass, all scheduling wheels corresponding to changing counter bits are serviced.

## 5.1.2. Fast Forward Mechanism

The binary counter used in the above implementation is increased by one at the start of a scheduling pass. However, if all scheduling wheels that are enabled in a given pass are empty, we must increment the counter again to find a queue from which to send. In the worst-case it may take many increment steps to find a non-empty queue and during these steps, link bandwidth may be lost. To avoid this, we introduce a *fast forward* mechanism for the counter.

The idea is to increment the counter with a carry-in at the position of the right-most non-empty scheduling wheel. We keep a mask register to indicate non-empty wheels that have not been served. We also keep a carry-in register with only one bit set at the position corresponding to the least significant '1' bit of the mask register. After each pass, the value the carry-in register is added to the counter. The resulting right-most changing bit always corresponds to a non-empty scheduling wheel. The fast forward algorithm is shown below.

```
Initially,
    PreviousCounter = 0;
    CurrentCounter = 0;
    Mask: Bit i is set to '1' if scheduling wheel i is non-empty, '0' if scheduling
        wheel i is empty;
Loop:
    CarryIn = Position of the least significant '1' bit of (Mask);
    PreviousCounter = CurrentCounter;
    CurrentCounter = CurrentCounter + CarryIn;
    ChangingBits = PreviousCounter XOR CurrentCounter;
    CurrentMask = Mask;
    While ((CurrentMask & ChangingBits) != 0)
        CurrentWheel = Position of the least significant '1' bit of (CurrentMask
                & ChangingBits);
        Serve all queues in the scheduling wheel CurrentWheel;
        CurrentMask[CurrentWheel] = 0;
        If (scheduling wheel CurrentWheel becomes empty )
            Mask[CurrentWheel] = 0;
        If (New queue is added to an empty scheduling wheel j)
            Mask[j] = 1;
```

The following example shows how the algorithm works. Table 5-1 gives the parameter values of the fast forward counter at the beginning of a pass. Table 5-2 shows the selection process.

**Table 5-1 Parameters of the Fast Forward Counter**

| *Current Counter* | *Previous Counter* | *Changing Bit* | *Mask* | *CarryIn* |
|---|---|---|---|---|
| 0100 | 0011 | 0111 | 1011 | 0001 |

**Table 5-2 Binary Scheduling Wheel Selection Process**

| *Current Counter* | *Current Mask* | *CurrentMask & ChangingBit* | *CurrentWheel* |
|---|---|---|---|
| 0100 | 1011 | 0011 | 0001 |
| 0100 | 1010 | 0010 | 0010 |
| 0100 | 1000 | 0000 | - |

The value of the mask register *Mask* is $(1011)_2$, which means only Wheel 2 is empty. So the carry-in register *CarryIn* has the value $(0001)_2$. The previous value of the counter is $(0011)_2$. The current value of the counter is equal to the sum of the previous value and the carry-in value, which is $(0100)_2$. The '1' bits in the register *ChangingBit* indicate the changing bits of the counter. *CurrentMask* is set to the value of *Mask* initially. (*CurrentMask & ChangingBit*) gives all scheduling wheels to be served in a pass. Since only one wheel can be visited at a time, *CurrentWheel* specifies the current scheduling wheel to be served. When a scheduling wheels is visited, all the queues on that wheel can send one cell to the outgoing link. After a scheduling wheel is served, the corresponding bit in *CurrentMask* is then cleared. This process continues until all selected scheduling wheels have been served once.

With the fast forward mechanism, the selection time becomes essentially independent of the total number of weights. While the time to select the least significant ‘1’ bit does require more than constant time, hardware implementation can easily be made fast enough that this does not becomes an issue for realistic values of *W*. Consequently, cells can be selected and forwarded in essentially constant time.

# 5.2. Per VC Packet Level Discarding Algorithm

End-to-end transport layer protocols send data in packets. These packets are further segmented into 48 bytes cells using AAL5 when data is transmitted through ATM networks. Since end-to-end error checking and retransmission are done on a packet basis, a single lost ATM cell can cause an entire transport layer packet to be discarded and retransmitted. Several packet level discard mechanisms have been proposed to maintain packet integrity. However, these mechanisms are based on FIFO queueing, where all virtual circuits share the same queue. In order to provide Quality-of-Service (QoS), per virtual circuit (VC) queueing is necessary in the presence of large buffers. New algorithms are needed for per VC queueing, to support QoS. The *Weighted Fair Goodput* (WFG) algorithm is designed for use with per VC queueing, and works with the cell scheduling algorithm to ensure that each VC receives the proper fraction of the link’s bandwidth during an overload period.

## 5.2.1. Existing Packet Level Discard Mechanisms

Transport layer protocols guarantee that all transport layer packets are delivered without error using retransmission. The ATM adaptation layer 5 (AAL5) segments transport layer packets into ATM cells. A single cell loss in a packet will cause the entire packet to be retransmitted. Therefore, if a cell is discarded by the network due to overload, transmitting the rest of the packet simply wastes bandwidth. Therefore, during the overload period, it is more important to maintain high *goodput*, where goodput is the fraction of the link's capacity that is used to transmit complete packets.

Packet Tail Discarding discards the rest of the packet if one cell in the packet is discarded [16]. This prevents the link bandwidth from being further wasted transmitting fragments of packets. However, since the first part of the packet will be discarded by the end-point eventually, the goodput can drop to zero when the link is really congested.

Early Packet Discarding (EPD) [16] can achieve 100% goodput if the link buffer is sufficiently large. The idea of Early Packet Discarding is to make a decision to propagate a packet or not at the packet boundary. Each virtual circuit has two states: active and inactive. Packets of an active virtual circuit are propagated, while packets of an inactive virtual circuit are dropped. A virtual circuit only changes states at packet boundaries based on the global buffer level. If the buffer does not overflow or become empty during an overload period, all the link bandwidth is used to propagate complete packets.

Early Packet Discard with Hysteresis (EPDH) [18] checks the rising and falling of the buffer level as well as the threshold when it comes to a packet boundary. This improves the performance of early packet discard when buffer space is limited.

Fairness is another issue that needs to be considered during the overload. Fair EPD with Hysteresis (FEPDH) [18] takes fairness into consideration. FEPDH uses two thresholds. At the packet boundary, it uses a similar algorithm to EPDH but also checks the fair share of the link if the buffer level is between the two threshold levels. This modification achieves some degree of fairness by keeping track of the number of cells that have been placed in the buffer. However, the hysteresis mechanism prevents the fairness heuristic from providing any real guarantee of fair treatment, since virtual circuits may miss the chance to change states if packet boundaries fall into the wrong part of the buffer occupancy curve.

### 5.2.2. Weighted Fair Goodput Algorithm

The above packet level discarding algorithms are based on FIFO queueing. In FIFO queueing, large buffers help avoid data loss during short overloads. However, when a link is overloaded for a long time, the combination of large buffers and FIFO queueing can lead to poor performance. Once the buffer fills, the cell loss rate is just a function of the input load. Cells in the buffer encounter large delays and it takes a long time for a large buffer to recover from overload. Therefore, systems that use FIFO queueing try to minimize buffer size.

In a system based on per VC queueing, large buffers do not impose large delays for VCs using only their allocated share of the link bandwidth. The system can have large buffers without causing well-behaved virtual circuits to have longer delays when the buffer is full. This makes the use of large buffers more attractive and allows the system to accommodate larger temporary data surges without loss.

However, ensuring that each VC receives its allocated bandwidth becomes important in order to provide QoS support. Existing algorithms cannot ensure this. The WFG algorithm can be used in conjunction with weighted scheduling schemes such as the Binary Scheduling Wheels algorithm to guarantee QoS. During an overload period, all virtual circuits forward cells according to their weight factors. The packet level discard mechanism keeps two states for each virtual circuit: active and inactive, and changes the states of virtual circuits at packet boundaries. The algorithm seeks to ensure that QoS requirements are satisfied by making virtual circuits inactive only when there are still enough cell’s in a virtual circuit’s queue to last until the start of the next packet. As a result, all virtual circuits are forwarded at their reserved rates and fairness is achieved. The Weighted Fair Goodput algorithm is shown below.

At a packet boundary for virtual circuit *V*, the WFG does the following:

> If the current global buffer level is above $b_h$, *V* is active and its queue length is more than $q_0$, make *V* inactive.
>
> If the current buffer level is above $b_{h,}$ V is inactive and its queue length is less than $q_0$, make *V* active.
>
> If the current buffer level is below $b_h$, and *V* is inactive, make *V* active.
>
> In all other cases do not change the state of *V*.

where $b_h$ is the threshold to avoid buffer overflow, and $q_0$ is the minimum queue length to guarantee no VC queue becomes empty before it is turned on again. The derivation of the lower bound for $q_0$ and minimum buffer size is shown below.

Let $q_0$ be the lower bound on the queue length that guarantees no VC queue becomes empty during inactive state. Let *n* be total number of VCs. Let $\delta$ be the fraction of VCs that send data at rates higher than their reserved rates. Let $\lambda$ be the input rate of those mis-behaving VCs. Let *w* be the reserved bandwidth of the mis-behaving VCs. Let *l* be the packet length.

Suppose the queue length of a VC is just below $q_0$ at the packet boundary and the VC is turned off. Figure 5-3 shows the changing queue length. The queue of the inactive VC drains at a slope of *w*. The maximum time to reach the next packet boundary is $l/\lambda$.

Therefore, in order to guarantee that no VC becomes empty before the next packet boundary, we must have $q_0 \geq \frac{w}{\lambda} l$. Because $w \leq \lambda$ by definition, we can satisfy this property in general by making $q_0 \geq l$. If the value of $\lambda$ is known, a smaller value for $q_0$ can be used, but typically this parameter is not available.

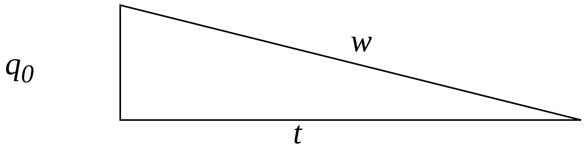


Figure 5-2 Change of Queue Length When Inactive

The minimum buffer size is derived below. In the worst case, a virtual circuit comes to its packet boundary when its queue length is just below $q_0$. Therefore, the virtual circuit will be in the active state. Figure 5-3 shows the change of the queue length in this case. The queue length increases at the rate of $d$, where $d = \lambda - w$. Since the time to reach the next packet boundary is $l/\lambda$, the total excursion above $b_0$ is $\frac{l}{\lambda}(\lambda - w)$ .

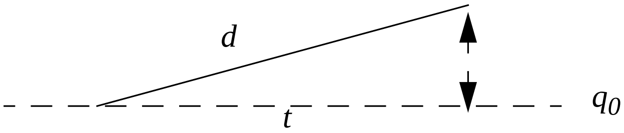


Figure 5-3 Change of Queue Length When Active

Assume that $q_0 = \frac{w}{\lambda} l$. The queue length increases at most $q_0 + \frac{l}{\lambda}(\lambda - w) = \frac{w}{\lambda} l + l - \frac{w}{\lambda} l = l$. Thus, total buffer size to avoid buffer overflow is $\delta n l$. If we have a buffer size of $\delta n l$ above the threshold, we can avoid buffer overflow entirely.

With a minimum buffer size derived above, all virtual circuits can be forwarded at their reserved rate during overload periods and 100% goodput can be achieved. The other interesting property of the WFG algorithm is that “well-behaved” virtual circuits sending at their allocated rates do not have any cell loss during the overload period. Because they are less likely to accumulate cells in the buffer (in the presence of appropriate queue scheduling methods), “well-behaved” virtual circuits are unlikely to be turned off during the overload period and never need packet retransmission. Virtual circuits sending at rates higher than their allocated shares are turned off during overloads. This causes packet loss, triggering end-to-end flow control mechanisms in transport protocols.

The hardware cost to implement the WFG algorithm is analyzed below. We need a queue length counter for each virtual circuit. For a buffer that can accommodate 1 million cells, a 20-bit counter is needed for each VC. The maximum transport layer packet length is limited by the Maximum Transfer Unit (MTU). In practice, the MTU is generally a few thousand bytes. Since an ATM cell has a payload of 48 bytes, a transport layer packet is usually segmented into 20 to 100 ATM cells.

## 5.3. Configuring Links with Excess Capacity

In the standard configuration, total external link bandwidth is equal to or less than the bandwidth of the Dynamic Queue Management Chip. In this case, scheduling techniques described in chapter 4 can be used. In particular, we need not make bandwidth that is not needed by one port, available to other ports.

However, it is possible to configure external links in such a way that the total link bandwidth is more than the bandwidth of the chip. In this case, if a selected port does not have cells destined for it, its cell cycles should be available to other links that have cells to send. Each virtual port can forward a cell every 16 cell times, and has bandwidth equivalent to 150 Mb/s. In an over configured case, we associate fewer virtual ports with each physical link. For example, instead of associating four virtual ports with an OC-12 link, we may associate just one virtual port with it. In this case, if all links are busy, the OC-12 link is only guaranteed to receive 150 Mb/s bandwidth. If enough of the links are idle, however, it is possible to allow the OC-12 link to use the full 600 Mb/s bandwidth. This takes advantage of the fact that not all links will use their full link capacity all the time, especially when traffic is bursty. Therefore, this configuration model allows links to send bursty traffic at higher rates if other links are not using their full bandwidth.

The output scheduling becomes more challenging in this case. First, when a selected link has no cell destined for it, the cell cycle should be made available to other links. Second, basic bandwidth allocations need to be guaranteed. Suppose a link is associated with *d* virtual ports, it should be able to use at least 150*d* Mb/s. Third, external links should never be allocated more bandwidth than they can use. Finally, the bandwidth not used to satisfy the basic bandwidth allocation of a link should be distributed in proportion to the number of virtual ports associated with the link. For example, if there is extra bandwidth available, an OC-12 link associated with two virtual ports should receive more bandwidth than the one associated with one virtual port.

A credit based scheduling scheme is proposed here. *Basic credit* and *extra credit* are used to handle the basic service and the distribution of unused bandwidth respectively. Basic credit is equal to the actual number of virtual ports that a link is associated with. Extra credit is defined as the number of additional cells a link may send every 16 cell time if there is excess bandwidth available. Extra credits are computed by subtracting the basic credits from the number of virtual ports listed in Table 4-2 on page 45. A credit table can be set up based on the link configurations. Table 5-3 shows how to set up the credit table for the configuration in Figure 5-4.

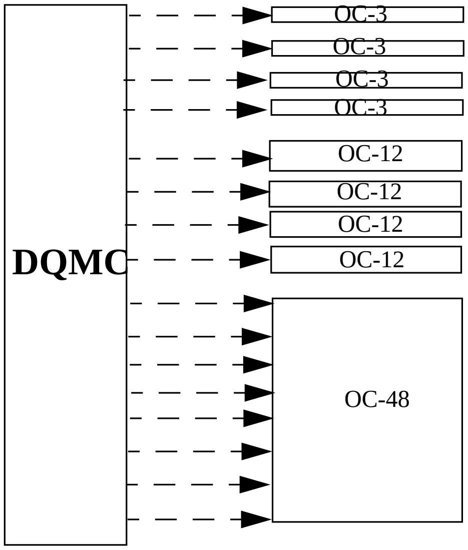


Figure 5-4 Example: Link Capacity Exceeding the DQM Bandwidth

As shown in the figure, each OC-3 and OC-12 link is associated with one virtual port. The OC-48 link is associated with 8 virtual ports. Based this configuration, each OC-3 link has one basic credit and zero extra credits. Each OC-12 link has one basic credit and 3 extra credits. The OC-48 link has eight basic credits and eight extra credits.

This credit table is reloaded every 16 cell times. Whenever a cell leaves based on basic credits, basic credits are decremented by 1. If there are no cells in the queue for a given link, the basic credits are also decremented by 1. Meanwhile, a link with extra credits can be scheduled to receive a cell. The extra credits for that particular link are decremented by 1 after receiving a cell.

Since the sum of the basic credits is no more than 16, all basic services can be satisfied. In addition, since the sum of the basic credits and extra credits of a particular link is equal to the maximum number of cells the link can receive without exceeding the link capacity, the link would never receive more cells than it can handle. In order to distribute unused bandwidth among competing links in proportion to their basic credits, the credit table needs to be modified as shown in Table 5-4.

**Table 5-3 Credit Table**

| Link Number | Link Type | Basic Credits | Extra Credits |
|---|---|---|---|
| 0 | OC-3 | 1 | 0 |
| 1 | OC-3 | 1 | 0 |
| 2 | OC-3 | 1 | 0 |
| 3 | OC-3 | 1 | 0 |
| 4 | OC-12 | 1 | 3 |
| 5 | OC-12 | 1 | 3 |
| 6 | OC-12 | 1 | 3 |
| 7 | OC-12 | 1 | 3 |
| 8 | OC-48 | 8 | 8 |
| 9 | - | 0 | 0 |
| 10 | - | 0 | 0 |
| 11 | - | 0 | 0 |
| 12 | - | 0 | 0 |
| 13 | - | 0 | 0 |
| 14 | - | 0 | 0 |
| 15 | - | 0 | 0 |

We use virtual ports to determine which link can receive a cell. A mask of four bits are used to convert the virtual port to the real link number. In the example above, if the port is any value between $(1000)_2$ and $(1111)_2$, the OC-48 link on output 8 can receive a cell. In the modified table, the basic credit is always 1 for each schedule number so that it need not be stored in memory. Extra credit is distributed among the virtual ports associated with the link. A non-empty bit indicates that there are cells currently destined for the link. When a virtual port is selected, one cell is allowed to forward to the link associated with it. In this way, the OC-48 link has eight chances to win an extra cycle as opposed to one chance for an OC-12 link every 16 cell times. Therefore, the extra bandwidth can be distributed in proportion to the basic credits.

**Table 5-4 Modified Credit Table**

| Virtual Port | Mask | Actual Link Number | Link Type | Basic Credits | Extra Credits | Non-Empty |
|---|---|---|---|---|---|---|
| 0000 | 1111 | 0 | OC-3 | 1 | 0 | 1 |
| 0001 | 1111 | 1 | OC-3 | 1 | 0 | 1 |
| 0010 | 1111 | 2 | OC-3 | 1 | 0 | 1 |
| 0011 | 1111 | 3 | OC-3 | 1 | 0 | 1 |
| 0100 | 1111 | 4 | OC-12 | 1 | 3 | 1 |
| 0101 | 1111 | 5 | OC-12 | 1 | 3 | 1 |
| 0110 | 1111 | 6 | OC-12 | 1 | 3 | 1 |
| 0111 | 1111 | 7 | OC-12 | 1 | 3 | 1 |
| 1000 | 1000 | 8 | OC-48 | 1 | 1 | 1 |
| 1001 | 1000 | 8 | - | 1 | 1 | 1 |
| 1010 | 1000 | 8 | - | 1 | 1 | 1 |
| 1011 | 1000 | 8 | - | 1 | 1 | 1 |
| 1100 | 1000 | 8 | - | 1 | 1 | 1 |
| 1101 | 1000 | 8 | - | 1 | 1 | 1 |
| 1110 | 1000 | 8 | - | 1 | 1 | 1 |
| 1111 | 1000 | 8 | - | 1 | 1 | 1 |

To select links without creating burstiness, a reverse order bit selection can be used for basic service scheduling. A token passing protocol can be used to elect a virtual port with non-zero credit and non-empty bit set. In this implementation, only one credit counter per virtual port is needed. The initial value is the sum of the basic credit and the extra credit. Figure 5-5 and Figure 5-6 shows the token passing circuitry.

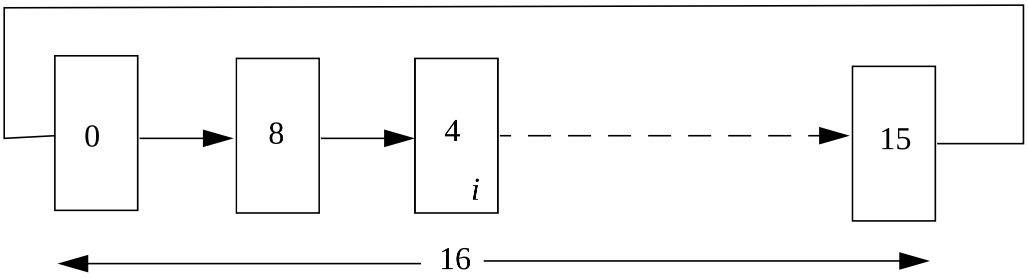


Figure 5-5  Token Passing Circuitry

.

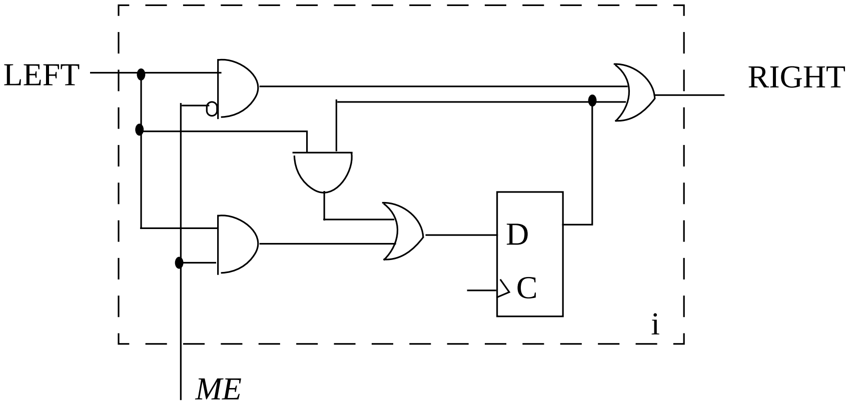


Figure 5-6  Internal Logic of Token Passing Circuit

Block *i* represents virtual port *i*. Blocks are connected as a ring in reversed-bit order. Only one virtual port can hold a token. The vitual port that has the token can forward a cell to the external link and decrement the credit counter by 1. It releases the token in the next cell time. When a token passes a virtual port that has extra credit and a cell to send, the virtual port can catch the token. If a token circulates and no virtual ports want it, the token needs to be held by the sender. The credit table is reloaded every 16 cell times. *ME* is enabled when a virtual port has credit and has cells to send. In order to initialize the table, sixteen bits are needed to specify the link configuration. The decoding can be done on chip and the table values can be computed based on the criteria described above.

# 6. DESIGN ANALYSIS

## 6.1. Performance Analysis

### 6.1.1. Overflow Probability

The DQM chip uses *Dynamic Queue Assignment* to allocate per VC queues without imposing restrictions on the choice of VPIs and VCIs. Dynamic Queue Assignment is implemented using set-associative memory (SAM) and content addressable memory (CAM). It is possible that too many virtual circuits are mapped to the same set in the SAM and have to spill over to the CAM. If the CAM is also full, the arriving cell on a new connection is discarded. This is referred as overflow. We need to understand how the configuration of the SAM and CAM affects the probability of cell loss due to overflows.

Let $n$ be the number of queues supported by the DQM. Let $\beta$ be the ratio of $n$ to the number of storage locations in the SAM. This quantity bounds the fraction of the SAM entries that can be in use at one time, and is called the *load factor*. Let $s$ be the number of entries in each set of the SAM, let $r = \frac{n}{\beta s}$ be the number of sets in the SAM and let $c$ be the number of entries in the overflow CAM.

We define the overflow probability to be the probability that when a cell *A* arrives on a "new connection" (one for which no queue is currently allocated), there is no available entry in either the SAM or the CAM.

To calculate the probability of overflow, we must make some assumption about the number and distribution of "in-use" entries in the set-associative memory and the CAM, at the time cell *A* arrives. We will assume that the "in-use" entries are randomly distributed in the following way. Let the set-associative memory and the CAM be empty initially. Now suppose that *n* cells arrive on *n* different virtual circuits. Assume that each of the arriving cells is equally likely to be mapped to any of the sets in the set-associative memory and that all arrivals are independent. Now, define $x_i$ to be the number of virtual circuits (out of the original *n*) that are mapped to set *i* in the set associative memory but spill over into the CAM.

$$Pr\{x_i = 0\} = \sum_{i=0}^{s} \binom{n}{i} \cdot \left(\frac{1}{r}\right)^{i} \cdot \left(1 - \frac{1}{r}\right)^{(n-i)}$$

and for $1 \le h \le n - s$,

$$Pr\{x_i = h\} = \binom{n}{s+h} \cdot \left(\frac{1}{r}\right)^{s+h} \cdot \left(1 - \frac{1}{r}\right)^{n-(s+h)}$$

Let *y* be the total number of virtual circuits that spill over from their sets in the set-associative memory to the CAM. Clearly $y = x_1 + \ldots + x_r$. We can get a conservative upper bound on *y* by treating the $x_i$ as independent random variables. In particular, if we let *z* be the random variable whose distribution is obtained by taking the convolution of the distributions for $x_1, \ldots, x_r$, then $Pr\{z \ge c\} \ge Pr\{y \ge c\}$.

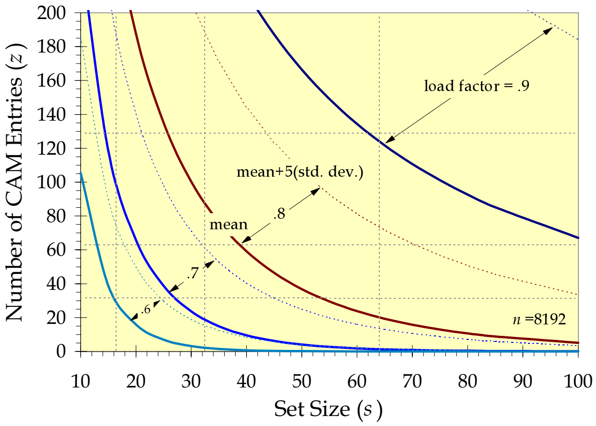


Figure 6-1 Recommended CAM Size

Figure 6-1 shows how $z$ varies with $s$ and the load factor $\beta$. The plot includes curves showing the mean value of $z$ and the mean plus five standard deviations. These were calculated numerically using the probability distribution of $x_i$ to obtain its mean and standard deviation, then multiplying these by $r$ and $\sqrt{r}$ to obtain the mean and standard deviation of $z$. These curves allow us to determine the values of $c$, $s$ and $\beta$ that will lead to good performance. For example, we can see that if we want to operate with a load factor of 0.8, then with a set size of $s$, we need a CAM size of at least 25 and more realistically, about 75 to keep overflows acceptably rare. If we want to operate with a higher load factor, we must increase $s$, $c$ or both. With a lower load factor, we can reduce $s$ and $c$, at the cost of more memory.

The overflow probability is no more than

$$\begin{aligned} Pr\{x_i > 0, z > c\} &= Pr\{x_i > 0\} Pr\{z > c | x_i > 0\} \\ &= Pr\{z > c\} Pr\{x_i > 0 | z > c\} \\ &\le min\left( Pr\{x_i > 0\}, Pr\{z > c\} \right) \end{aligned}$$

By the central limit theorem, we can estimate $z$ using a normal distribution. Let $\mu$ and $\sigma$ denote the mean and standard deviation of $x_i$. For any positive number $\gamma$,

$$\begin{aligned} Pr\{z > r\mu + \gamma \cdot \sigma\sqrt{r}\} &\approx \frac{1}{\sqrt{2\pi}}\int_{\gamma}^{\infty} e^{-\gamma^2/2}\,dx \\ &< \frac{1}{\gamma\sqrt{2\pi}} \cdot e^{-\gamma^2/2} \end{aligned}$$

Figure 6-2 plots the value of this last quantity, as a function of load factor, for several different choices of $s$ and $c$. With a set size and CAM size of 32, the estimated overflow probability is less than one in a million when the number of storage locations in the SAM is (1/0.65)$n$. If both are increased to 64, a load factor of nearly 0.8 yields the same overflow probability. For $n = 8192$, a load factor of 0.8 implies 10,240 SAM entries. With $s = 64$, this implies $r = 160$.

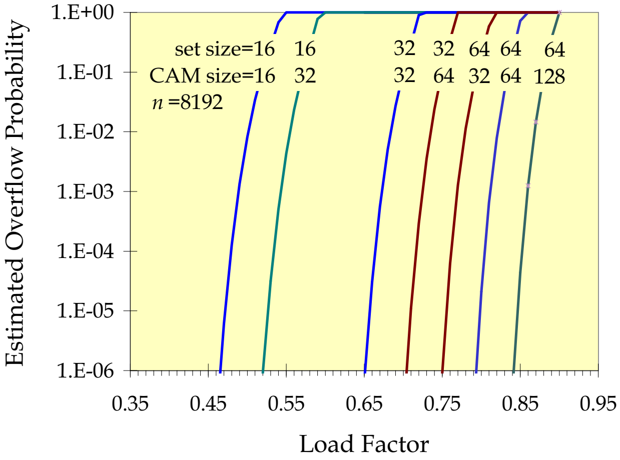


Figure 6-2 Overflow Probability

## 6.1.2. Overbooking of Queue Data Structures

Because the DQM assigns queues to virtual circuits dynamically, it is possible to support a larger number of virtual circuits than could be supported if queues were statically bound to specific virtual circuit identifiers. That is, we can overbook the DQM's data structures, taking a risk that on occasion we will not have an available queue to handle an arriving cell, forcing the cell to be discarded. In order to exploit the potential for overbooking, it is important to understand how many virtual circuits can be supported with a given number of queues. Here, we make some basic observations, leaving a detailed analysis of overbooking to a future study.

Note first that if a DQM supports *n* queues, there will always be an available queue if the number of queued cells is $\leq n$. For non-bursty traffic, the queue length rarely exceeds even 100 cells for traffic loads of 95% or less. Thus, for $n = 8192$ , the probability is exceedingly small that an arriving cell will not find an available queue, even if the number of virtual circuits using the link is over one million.

For bursty traffic, it is also possible to overbook the queues extensively. Suppose we have *m* identical independent on-off bursty sources with $m > n$ and an average time of *T* between the start of successive bursts (from any single source). If the input traffic (averaged over periods longer than *T*) is less than the link rate, then the average rate from each individual source is the link rate divided by *m*, which is small if *n* is reasonably large. Typical virtual circuits have peak rates of perhaps 20 times the average rate. For $n = 8192$ , this results in virtual circuit peak rates that are less than 0.25% of the link rate. For such traffic, the queue rarely accumulates a significant backlog of cells, so again, an arriving cell will generally find an available queue.

Suppose however, that we have sources with peak rates that are much larger than their average rates. In particular, assume that bursts arrive independently and instantaneously, with an exponentially distributed time between bursts from any specific source. Also, assume that burst lengths are exponentially distributed and that each burst is assigned a separate queue (even two bursts coming from the same source), and that all non-empty queues are drained at a rate that is inversely proportional to the number of non-empty queues (modeling a round-robin queue scheduler). This queueing system can be modeled by a birth-and-death process, in which the state index corresponds to the number of non-empty queues. If we let the number of sources go to infinity, while keeping the time between successive burst arrivals constant, this birth-and-death process becomes identical to that for the M/M/1 queue. This implies (among other things) that the probability that there are more than $j$ non-empty queues is $\rho^{j+1}$, where $\rho$ is the normalized traffic intensity for the queueing system. For $\rho = 0.95$, this probability is less than $10^{-6}$ for all $j > 268$. These results show that the DQM queues can be overbooked by a large factor, if $n$ is sufficiently large. With smaller $n$, the potential for overbooking is reduced somewhat, but even with as few as 1024, we are unlikely to run out of queues under any realistic traffic conditions.

# 6.2. Cost Estimation

## 6.2.1. On-Chip Memory

The complexity of the chip is mainly driven by the on-chip memory. Let $n$ be the number of queues supported by the DQM. Let $B$ be the number of cell slots in the Cell Store. Let $\beta$ be the load factor in the SAM. Let $W$ be the number of weighted levels. Let $m$ be the number of outputs. The memory requirements are estimated below.

## Queue Selector

The memory required for the Queue Selector is $(2n\log n)/\beta + n\log n + n\log(n/\beta)$ as shown in Figure 6-3. The calculation includes the SAM, the Free Queue List and the Address Map. For $n = 8192$, a load factor of 0.8 gives a memory requirement of about 60 Kbytes, while a load factor of 0.5 gives a memory requirement of 80 Kbytes. Note that the load factor affects only the SAM, but not the Free Queue List or the Address Map. When the load factor is 0.5, the memory required for the SAM is 1/3 of the total.

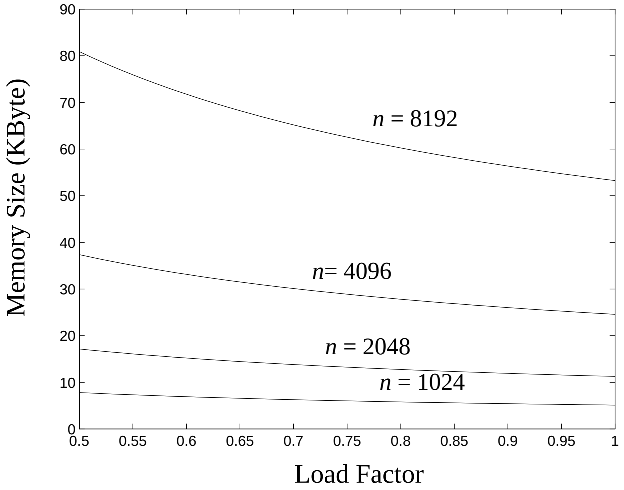


Figure 6-3 Memory Size for the Queue Selector

Figure 6-4 shows the memory area estimation for a 0.35 micron CMOS process. Combining the results in Figure 6-3 and Figure 6-4, with 1024 queues and a load factor of 0.8, the chip area consumed by dynamic queue assignment is less than $2\ \mathrm{mm}^2$. For 8192 queues, the area is approximately $17\ \mathrm{mm}^2$, or less than 20% of the area of a $100\ \mathrm{mm}^2$ chip. The analysis indicates that the number of queues could be increased to 16K without consuming an excessive fraction of the chip area. 64K queues can be supported in a 0.18 micron process.

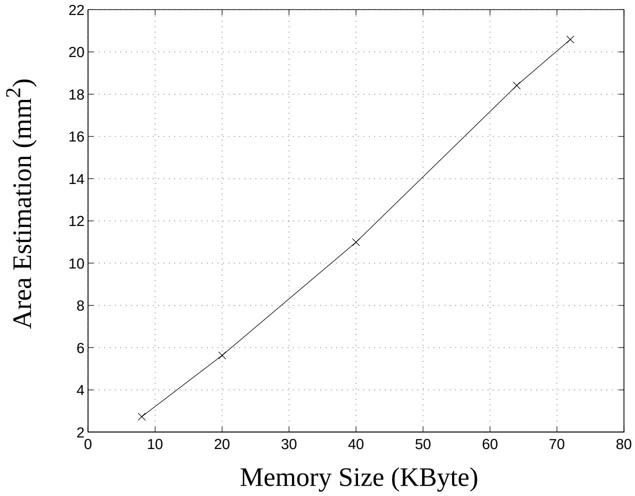


Figure 6-4  Memory Area Estimation (0.35 Micron)

## Queue Manager

The Queue Manager maintains a queue list of *n* entries. Virtual path queues are ignored here. Each entry contains two pointers to the cell buffer in the external memory. The memory size is $2n\log B$. Figure 6-5 shows the memory required for the Queue Manager. Because the memory size does not increase significantly, a cell buffer of one million cells (corresponding to 20 bits in address) is a reasonable choice for the DQM chip.

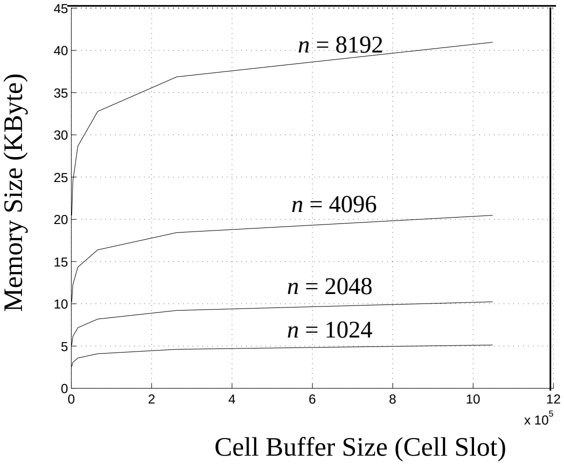


Figure 6-5  Memory Size for the Queue Manager

## Output Scheduler

The Output Scheduler implements either two priority scheduling or weighted round-robin scheduling using Binary Scheduling Wheels (BSW) algorithm. Since two priority scheduling has the same structure as BSW with $W = 2$, we only calculate the memory required by the BSW algorithm. Figure 6-6 shows the structures in the Output Scheduler that implement the BSW algorithm.

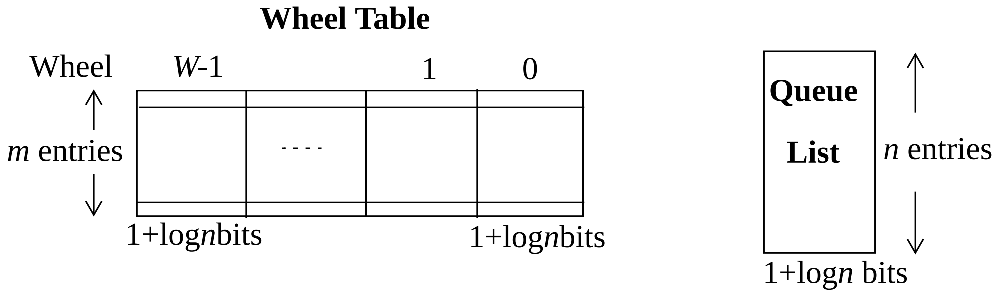


Figure 6-6 Structures in the Output Scheduler

At each output, we construct *W* scheduling wheels. We need a queue list with *n* entries to represent the queues in scheduling wheels. Since a virtual circuit has only one destination, a queue can only be placed on one of the scheduling wheels. Therefore, the queue list can be shared by all output wheels. The actual scheduling wheels are constructed by linking corresponding entries in the queue list. An additional bit is added to entries in the queue list to distinguish the first queue in a wheel. To access the scheduling wheels, a wheel table is used to store the pointers to the scheduling wheels. The memory requirement for the BSW for *m* outputs is $(n + mW)\ (1 + \lceil \log n \rceil)$ .

Figure 6-7 shows the memory size for various parameters with $m = 16$. Note that only the wheel table depends on the number of weights. The increment in memory size to implement 64 weights over a simple two priority design is less than 2 KBytes. 32 distinct power of 2 weights are sufficient to specify bandwidths ranging from 2.4 Gb/s to less than one bit per second. Even with 8192 virtual circuits in the system, the total memory requirement for the scheduler is less than 15 KBytes. Therefore, the BSW algorithm implements the weighted round-robin scheduling in a very efficient way.

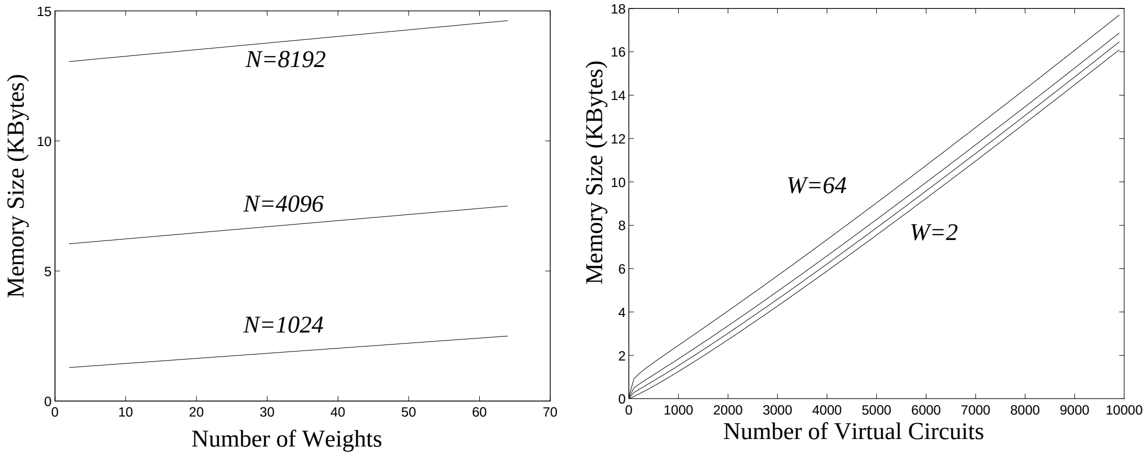


Figure 6-7 Memory Required for Cell Scheduling

Because a VC queue only belongs to one scheduling wheel, adding and removing a queue from a scheduling wheel can be done in constant time. This is the major reason for restricting weights to be powers of 2. The algorithm can be extended to more general weights by allowing each queue to appear in multiple wheels. If a queue can appear in *j* wheels, the ratio between successive weights is $\frac{2^j}{2^j - 1}$, but both the scheduling time and the size of the queue list increase by a factor of *j*.

## Free Slot Manager

The Free Slot Manager maintains a on-chip free slot cache. Assume that the cell buffer can store a maximum of one million cells. A free slot cache with 64 entries is 0.2 Kbyes.

## Output Master

The Output Master has four FIFO queues, each of which can contain 8 cells. The memory size is 2.2 KBytes.

### Total Memory and Area Estimates

Memory and area estimates for the entire chip is listed in Table 6-1. The parameters are $n = 8192$, $\beta = 0.8$, $W = 32$, $B = 2^{20}$, and $m = 16$. Area is estimated using typical data for $0.35\ \mu m$ 3 metal layer CMOS process.

**Table 6-1 On-Chip Memory Estimates**

| Block Name | Size (KB) | Size (%) | Area (KB) | Area (%) |
|---|---|---|---|---|
| Queue Selector | 60 KB | 52% | 17 mm$^2$ | 51% |
| Queue Manager | 40 KB | 34% | 11 mm$^2$ | 33% |
| Output Scheduler | 14 KB | 12% | 4 mm$^2$ | 12% |
| Free Slot Manager | 0.2 KB | 0.1% | 0.18 mm$^2$ | 0.1% |
| Output Master | 1.8 KB | 2% | 1.32 mm$^2$ | 4% |
| Total | 116 KB | 100% | 33.5 mm$^2$ | 100% |

## 6.2.2. External Memory

The Cell Store and the Free Slot List are stored in the external memory. The Dynamic Queue Management Chip supports up to $2^{20}$ cell slots. The maximum size of the Cell Store is 60 MBytes. The Free Slot List stores all the pointers to the Cell Store. If the maximum size of Cell Store is supported, the size of the Free Slot List is $20 \times 2^{20} = 2.5$ MBytes. Therefore, the maximum memory size of the system is 62.5 MBytes. This gives the potential for growth as the price of memory drops, making very large buffers affordable. In the short term, smaller buffers (1 - 4 MBytes) are more realistic, given current static RAM costs of $75 -100 per MByte.

Within each cell time of 116 ns, a cell is written into the Cell Store, and a cell is read out of the Cell Store. An entire read cycle or write cycle is at most 58 ns. We also need to limit the data path because of the limitation of the pin count. Therefore, SRAM is a proper choice at the current time.

The data path to the external memory is 160 bits wide. Since the size of a cell is 448 bits, and the pointer field stored with the cell is 20 bits, three memory accesses are needed to write a cell to the Cell Store or to read a cell from the Cell Store. In other words, during every cell time, there are three memory reads and three memory writes.

The suggested memory module is Micron’s 128K x 32 SYNCBURST SRAM with 11 ns cycle time. Each cell time contains 7 memory cycles. Three cycles are used for writing a cell; one cycle is for switching from write to read, with another three cycles for reading a cell.

Since the internal free slot cache is used to manage the free slots, the Free Slot List only needs to be accessed when there is no cell arriving or leaving for a long time. A write cycle can be used to access the Free Slot List in the first case and a read cycle can be used in the second case. As a result, the Free Slot List is accessed when there is an idle read or write cycle. Therefore, there is no extra cost to maintain the Free Slot List. Table 6-2 summarizes the requirements of the external memory.

**Table 6-2 External Memory Estimates**

| | |
|---|---|
| Data Path | 160 bits |
| Maximum Memory Size Supported | 62.5 MBytes |
| Suggested Memory Size | 4 MBytes |
| Memory Operations per Cell Time | 3 reads, 3 writes |
| Suggested Memory Module | Micron 128K*32 SRAM |
| Cycle Time | 11 ns |
| Total Number of Chips | 10 |

## 6.2.3. Gate Estimation

In order to estimate the total chip area, we need to estimate the number of equivalent gates in the chip. We count the number of flip-flops and multiply it by two in order to include the gates in the control logic. The area estimates are based on a typical 0.35 $\mu m$ CMOS process with three metal layers. Table 6-3 shows the gate and area estimation.

1 flip-flop with reset = 5 nand gates = $91 um^2$

Circuit area after routing = 5 * (Area of all gates)

**Table 6-3 Gate Estimates**

| Component | Number of Flip-Flops | Equivalent Gates | Area Before Routing | Area After Routing | Percentage of Area |
|---|---|---|---|---|---|
| Input Master | 100 | 500 | 0.0455 $mm^2$ | 0.223 $mm^2$ | 4.1% |
| Memory Controller | 1200 | 6000 | 0.545 $mm^2$ | 2.725 $mm^2$ | 49.9% |
| Output Master | 300 | 1500 | 0.137 $\mathrm{mm}^2$ | 0.685 $\mathrm{mm}^2$ | 12.5% |
| Queue Selector | 300 | 1500 | 0.137 $\mathrm{mm}^2$ | 0.685 $\mathrm{mm}^2$ | 12.5% |
| Queue Manager | 250 | 750 | 0.068 $\mathrm{mm}^2$ | 0.340 $\mathrm{mm}^2$ | 6.2% |
| Output Scheduler | 100 | 500 | 0.046 $\mathrm{mm}^2$ | 0.230 $\mathrm{mm}^2$ | 4.2% |
| Free Slot Manager | 150 | 750 | 0.068 $\mathrm{mm}^2$ | 0.340 $\mathrm{mm}^2$ | 6.2% |
| Total | 2400 | 12000 | 1.092 $\mathrm{mm}^2$ | 5.460 $\mathrm{mm}^2$ | 100% |

## 6.2.4. Pin Count

The total pin count of the chip shown in Table 6-4.

The chip needs a package of 400 pins. If we want to reduce the pin count, a faster external SRAM can be used. This is achievable with the price drop on fast components.

**Table 6-4 Pin Counts**

| Name | Pin Counts |
|---|---|
| Data Pins | 256 |
| Control Pins | 100 |
| Vcc | 22 |
| GND | 22 |
| Total | 400 |

## 6.2.5. Total Chip Area and Physical Packaging

The total area of the chip is $33.5 + 5.5 = 39\ mm^2$. Therefore, the estimated size of the DQM chip is less than 7 mm x 7 mm. The chip can use a 411-pin ceramic PGA package with dimension of 2 inch by 2 inch.

The Micron 128Kx32 SRAM is available in either a 100-pin TQFP (Thin Quad Flat Package) or a 119-Bump BGA (Ball Grid Array). The dimension for the BGA package is 0.89 inch by 0.57 inch (including mold protrusion).

The DQM chip with a 4 MByte external memory configuration indicates a 10 $\text{inch}^2$ space on a PC board.

# 7. CONCLUSIONS

The goal of this thesis is to explore critical issues in providing Quality-of-Service in ATM networks and give a detailed design of a Dynamic Queue Management chip that can help provide effective QoS.

The research contributions of the thesis are listed below:

**- Dynamic Queue Assignment**

The chip implements per VC queueing using dynamic queue assignment, which avoids restricting the choice of VPIs and VCIs unduly. Dynamic queue assignment makes the cost of the system proportional to the number of VPI/VCIs actually used, rather than the maximum possible number that could be used.

**- Unlimited Buffer Scaling**

The DQM chip is designed in such a way that the cell buffer can be scaled up without increasing the chip complexity significantly. Both the cell buffer and the information to maintain the cell buffer (including the free slot list) are stored in the external memory. The only effect of increasing the buffer size is that the number of bits to address the cell buffer will increase. So the chip needs to maintain a few more bits for each queue if the buffer size increases.

**- Internal Free Slot Caching**

The internal free slot recycling mechanism used in the DQM chip allows the free slot list to be in the external memory without increasing the memory bandwidth requirements. Without this mechanism, the bandwidth of the external memory would need to be high enough to handle both the free slot list and the cell buffer without suspending an output cycle. However, with the internal free slot recycling, the on-chip manager only needs to access the free slot list when there is no cell coming in for a long time or there is no cell going out for a long time. In either case, there is an idle write cycle or an idle read cycle for the cell buffer. The free slot list in the external memory can be accessed using these idle memory cycles. Therefore, the memory bandwidth only needs to be sufficient for the cell buffer, which makes the timing a lot easier.

**- Cost Efficient Fair Queueing**

Multiple binary priority classes can be supported using the Binary Scheduling Wheels algorithm. This algorithm allows priorities to be specified as a power of 2. VC queues with binary weights are placed on one of the scheduling wheels. Unlike simple weighted round-robin, the Binary Scheduling Wheels spread out the traffic destined for the same output in order to prevent congestion at the link interface. A fast forward counter with the ability to skip empty scheduling wheels makes the algorithm scalable. The implementation of the Binary Scheduling Wheels algorithm requires only a small increment in the hardware cost.

**- Flexible Output Link Configuration**

The DQM chip allows up to 16 external links of various rates to share one output port of the WUGS operating at 2.4 Gb/s. The external links can be configured as any combination of OC-3, OC-12, G-link and OC-48 links. The idea of virtual ports has been introduced to define various configurations. One physical link is normally associated with one or several consecutive virtual ports. In order to avoid cell accumulations in the link interface, a bit reversal mapping is used to distribute the selection of virtual ports. The DQM chip allows the external links to be over configured, where the total bandwidth of the external links exceeds the bandwidth of an output port of the WUGS. This takes the advantage of the fact that not all links are busy at the same time and allows bursty traffic to be sent at a higher rate using statistical multiplexing among external links. An over configured link may use its full bandwidth if enough other links do not use their basic bandwidth. A novel credit-based scheduling mechanism and a token passing arbitration circuit are used to select virtual ports. The algorithm ensures the basic bandwidth that an external link subscribes to. An over configured link may use extra bandwidth made available to it, but can never be allocated more bandwidth than it can use.

**- Per VC based Packet Level Discarding**

Because the transport layer protocol transmits packets, a single cell lost in the ATM layer will cause the entire transport layer packet to be retransmitted. Therefore, it is very important to maintain high goodput during the overload period. A lot of packet level discarding mechanisms have been discussed in the context of FIFO queueing. However, no work has been shown for the systems that support per VC queueing, especially when fairness needs to be considered. The major difference between these two systems is that larger buffers do not impose large delay penalties in the per VC queueing systems during overload periods. The major concern in a per VC queueing system is how to preserve fairness as well as the packet integrity during the overload period. This thesis shows how to solve this problem using the Weighted Fair Goodput (WFG) algorithm designed especially for systems that use per VC queueing. All virtual circuits can forward cells at their reserved rates during the overload period using WFG.

There are several questions that need to be further explored.

First, simulations and detailed analysis on the Weighted Fair Goodput algorithm will help us learn more about the packet level discarding with per VC queueing. We are seeking a better algorithm with reduced buffer requirements and fair share of buffer space during overload period.

Second, the multiple priority classes with binary weights can be extended to arbitrary weights using almost the same structure as the Binary Scheduling Wheels. This is done by placing a VC queue with arbitrary weights on several binary scheduling wheels instead of one. This extension uses the fact that an arbitrary number can be decomposed as the sum of power of 2 terms. One thing that is not clear here, is how to add and remove queues to/from multiple scheduling wheels in an efficient way.

Third, we need to understand more about overbooking per channel data structures in order to take advantage of dynamic queue assignment. A detailed analysis on overload probability and a simulation will be helpful.

# APPENDIX A. SIGNAL DEFINITIONS

## 1. INPUT MASTER (IMST)

### SIGNALS FROM OPP

**DATA_OPP -** Cell data received from OPP.

### SIGNALS TO QSEL

**VPI_IMST -** This signal is the virtual path identifier of the incoming cell. The IMST copies it from the VPI field of the cell format.

**VCI_IMST -** This signal is the virtual circuit identifier of the incoming cell. The IMST copies it from the VCI field of the cell format.

**TYPE_IMST -** This 1 bit signal is copied from the VPT field of the cell. This signal is 1 if the cell belongs to a virtual circuit connection, 0 if the cell belongs to a virtual path connection.

**PRI_IMST -** This 1 bit signal is 1 for high priority and 0 for low priority. The IMST copies it from the CS field of the incoming cell.

**WT_IMST** - This signal is copied from the lower 2 bits of WT field.

**A5_IMST -** This 1 bit signal is copied from the A5 field.

**U_IMST -** This 1 bit signal is copied from the u-bit in the PT field.

### SIGNALS TO MCTRL

**DATA_IMST -** This 32 bit signal carries the cell data passed to the MCTRL.

## 2. Queue Selector (QSEL)

### SIGNALS FROM IMST

**VPI_IMST -** This signal is the virtual path identifier sent from the IMST. The virtual path identifier (VPI) field is part of the cell format. The IMST obtains the VPI from the incoming cell and sends it to the QSEL.

**VCI_IMST -** This signal is the virtual circuit identifier sent from the IMST. The virtual circuit identifier field is also obtained from the incoming cell by the IMST.

**TYPE_IMST -** This 1 bit signal indicates if the incoming cell belongs to a virtual path connection or a virtual circuit connection. This bit is copied from the VPT field of the cell by the IMST. This signal is 1 if the cell belongs to a virtual circuit connection, 0 if the cell belongs to a virtual path connection.

**PRI_IMST -** This 1 bit signal indicates the priority of the incoming cell. If the cell is of high priority, this bit is 1. Otherwise, the bit is 0. The IMST copies the PRI signal from the CS field of the incoming cell.

**WT_IMST** - This signal indicates the weight of the cell. It is passed to WT_QSEL without modification.

**A5_IMST -** This bit indicates the cell belongs to an AAL5 packet.

**U_IMST -** This bit indicates the last cell in an AAL5 packet. It is used for packet level discarding.

### SIGNALS FROM QMGR

**DEL_QMGR -** This signal is set to 1 if the queue is empty and needs to be removed from the QSEL. The Queue Manager sets this signal if the FIRST field and LAST field equal after a cell departs.

**DELQID_QMGR -** This signal carries the queue identifier of the queue to be removed. If the queue is empty, the QMGR sends the DELQID to the QSEL.

## SIGNALS TO QMGR

**PRI_QSEL -** This 1 bit signal is copied by the QSEL from the PRI_IMST. The QSEL does not use the PRI internally.

**TYPE_QSEL -** This signal is copied from the TYPE_IMST to indicate a VP or VC connection.

**OUT_QSEL -** This 4 bit signal carries the output number the incoming cell is destined for. The QSEL generates this signal using high order four bits of VPI.

**WT_QSEL -** This signal is copied from WT_IMST.

**QID_QSEL -** This signal provides the queue identifier of the incoming cell.

**A5_QSEL -** This signal is copied from A5_IMST.

**U_QSEL -** This signal is copied from U_IMST.

# 3. QUEUE MANAGER (QMGR)

## SIGNALS TO QSEL

**DEL_QMGR -** This signal is set to 1 if the queue is empty and needs to be removed from the QSEL. The Queue Manager sets this signal if the FIRST field and LAST field of the queue that outputs a cell equal.

**DELQID_QMGR -** This signal carries the queue identifier of the queue to be removed. If the queue is empty, the QMGR sends the DELQID with the DEL signal to the QSEL.

## SIGNALS FROM QSEL

**PRI_QSEL -** This 1 bit signal is copied to the NEWPRI_QMGR and passed to the OSCHL if the cell belongs to a new connection. The QMGR does not use the PRI internally.

**OUT_QSEL -** This 4 bit signal is copied to the NEWOUT_QMGR and passed to the QSCHL for a new connection.

**WT_QSEL** - This signal is copied to WT_QMGR.

**QID_QSEL -** This signal provides the queue identifier of the incoming cell. The Queue Manager uses it as an index to its queue list to get the value of the FIRST field corresponding to the slot in which the incoming cell is to be stored. The QMGR also copies this signal to the NEWQID_QMGR if it is a new connection. The QSEL passes NEWVC_QSEL to indicate a new VC connection. However, for a VP connection, the QMGR needs to compare the FIRST and LAST fields to detect a new connection.

## SIGNALS TO FSMGR

**RTNSLT_QMGR -** This signal carries the free slot to be returned to the FSMGR. The QMGR returns a slot when the cell in that slot departs.

**RTNEN_QMGR -** This signal is set to 1 if the QMGR returns a free slot.

**RQS_QMGR -** This signal is set to 1 if the QMGR requests a free slot from the FSMGR. The QMGR sends a request when it assigns a slot to a incoming cell.

## SIGNALS FROM FSMGR

**FSLT_FSMGR -** This signal carries a free slot to the QMGR.

**FSLTEN_FSMGR -** This signal is 1 if the FSLT_FSMGR is valid.

## SIGNALS TO OSCHL

**NEW_QMGR -** This signal is set to 1 if the incoming cell belongs to a new connection. If FIRST and LAST fields of the queue that an incoming cell is appended to are equal, it is considered as a new connection.

**NEWQID_QMGR -** This signal provides the queue identifier of a new queue to be added to the priority lists in the QSCHL. It is copied from the QID_QSEL for a new connection. It is valid only if the NEW_QMGR is 1.

**NEWOUT_QMGR -** This 4 bit signal indicates the output number of a new connection. It is copied from OUT_QSEL and is only valid when NEW_QMGR is 1.

**NEWPRI_QMGR -** This 1 bit signal specifies the priority of a connection. It is copied from PRI_QSEL for a new connection.

**NEWWT_QMGR -** This signal is copied from WT_QSEL for a new connection.

**EMP_QMGR -** This signal is set to 1 if a queue is empty and needs to be removed from the priority list in the OSCHL. It always has the same value as DEL_QMGR.

## SIGNALS FROM OSCHL

**NXTQID_OSCHL -** This signal provides the queue identifier of the queue to send a cell from.

**NXTEN_OSCHL -** This signal indicates that the NXTQID_OSCHL is valid.

## SIGNALS TO MCTRL

**RSLT_QMGR -** This signal provides the slot number of the cell being read in the CSTR. It is obtained from the FIRST field indexed by the queue identifier sent from the QSEL.

**RSLTEN_QMGR -** This signal indicates that the slot number for the departing cell is valid.

**WSLT_QMGR -** This signal provides the slot number for the incoming cell to write to. It is obtained from the LAST field indexed by the queue identifier sent from the OSCHL.

**WSLTEN_QMGR-** This signal is 1 when the WSLT_QMGR is valid.

**NXTPTR_QMGR -** This signal carries the pointer field to be written into the CSTR. The QMGR always keeps a free slot as the pointer field of the next arriving cell. The pointer is also used to update the LAST field. After assigning the free slot to the cell, the QMGR send a request (RQS_QMGR) to the FSMGR for another free slot.

### SIGNALS FROM MCTRL

**PTR_MCTRL -** This signal provides the pointer field of the departing cell. It is used to update the FIRST field of queue from which the cell departs.

**PTREN_MCTRL -** This signal is 1 if the PTR_MCTRL is valid.

## 4. OUTPUT SCHEDULER (OSCHL)

### SIGNALS FROM QMGR

**NEW_QMGR** - This signal is 1 for a new connection. The OSCHL needs to put the new queue on its priority list.

**NEWQID_QMGR -** This signal provides the queue identifier of a new queue to be added to the priority lists in the QSCHL. It is valid only if the NEW_QMGR is 1.

**NEWOUT_QMGR -** This 4 bit signal indicates the output number of a new connection. The OSCHL put the queue on the circular list of this particular output. It is only valid when NEW_QMGR is 1.

**NEWPRI_QMGR -** This 1 bit signal specifies the priority of a new queue. The OSCHL places the queue on the high priority list if NEWPRI is 1, and places it on the low priority list otherwise.

**NEWWT_QMGR** - This signal is the weight of a connection.

**EMP_QMGR -** This signal indicates that the queue that the OSCHL scheduled has becomes empty after sending a cell and needs to be removed from the priority list.

### SIGNALS TO QMGR

**NXTQID_OSCHL -** This signal provides the queue identifier of the queue that is selected to forward a cell to the output link.

**NXTEN_OSCHL -** This signal is set to 1 if the NXTQID_OSCHL is valid.

### SIGNALS TO OMST

**NXTOUT_OSCHL -** This 4 bit signal provides the output number that the departing cell is destined for.

**OUTEN_OSCHL -** This bit is set to 1 if the NXTOUT_OSCHL is valid and a cell is departing.

## 5. FREE SLOT MANAGER (FSMGR)

### SIGNALS FROM QMGR

**RTNSLT_QMGR** - This signal carries the free slot to be returned. The FSMGR recycles free slots internally. The returned slot is placed in its internal slot cache.

**RTNEN_QMGR -** This signal is 1 if the RTNSLT_QMGR is valid.

**RQS_QMGR -** This signal is 1 if the QMGR requests another free slot.

### SIGNALS TO QMGR

**FSLT_FSMGR** - This signal carries a free slot to the QMGR. The FSMGR sends a free slot to the QMGR when it receives the RQS_QMGR.

**FSLTEN_FSMGR -** This signal is set to 1 if the FSLT_FSMGR is valid.

### SIGNALS TO MCTRL

**HEAD_FSMGR -** This signal provides the address of the first free slot in the FSL.

**HDEN_FSMGR -** This signal indicates that the HEAD_FSMGR is valid. Another set of free slots need to be read out of the FSL.

**TAIL_FSMGR -** This signal provides the address in the FSL to return a set of free slots.

**TLEN_FSMGR -** This signal is set to 1 when a set of free slots are returned to the FSL. The TAIL_FSMGR and the WSLTS_FSMGR are valid only if TLEN_FSMGR is 1.

**WFSLTS_FSMGR -** This signal carries a set of free slots to be returned to the FSL. It is valid when the TLEN_FSMGR is set to 1.

### SIGNALS FROM MCTRL

**RFSLTS_MCTRL -** This signal carries a set of free slots read out of the FSL. The FSMGR needs puts them into the slot cache.

**RFSLTSEN_MCTRL -** This signal is 1 if the RFSLTS_MCTRL is valid.

## 6. Memory Controller (MCTRL)

### SIGNALS FROM QMGR

**RSLT_QMGR -** This signal provides the slot number of the departing cell in the CSTR. The MCTRL internally converts it to the external memory address.

**RSLTEN_QMGR -** This signal indicates that the RSLT_QMGR is valid. The MCTRL starts a read cycle if the signal is 1.

**WSLT_QMGR -** This signal provides the slot number for the incoming cell to write to. The MCTRL converts it to the external memory address internally.

**WSLTEN_QMGR-** This signal is 1 when the WSLT_QMGR is valid.

**NXTPTR_QMGR -** This signal carries the pointer field to be written into the CSTR. The MCTRL buffers the data and writes it to the CSTR with the cell.

## SIGNALS TO QMGR

**PTR_MCTRL -** This signal provides the pointer field of the departing cell. The MCTRL buffers the data read out from the CSTR first, then sends it to the QMGR.

**PTREN_MCTRL -** This signal is set to 1 if the PTR_MCTRL is valid.

## SIGNALS FROM FSMGR

**HEAD_FSMGR -** This signal provides the address of the first free slot in the FSL.The MCTRL converts it to the external memory address.

**HDEN_FSMGR -** This signal indicates that the HEAD_FSMGR is valid. The MCTRL accepts the HEAD_FSMGR and schedules a memory read from the FSL.

**TAIL_FSMGR -** This signal provides the address in the FSL to return a set of free slots. The MCTRL converts it to the external memory address.

**TLEN_FSMGR -** This signal is 1 when the TAIL_FSMGR and the WFSLTS_FSMGR are valid. The MCTR schedules a memory write to the FSL.

**WFSLTS_FSMGR -** This signal carries a set of free slots to be returned to the FSL. The MCTRL accepts it the TAILEN_FSMGR is set to 1.

## SIGNALS TO FSMGR

**RFSLTS_MCTRL -** This signal carries a set of free slot read out of the FSL. The MCTRL buffers the data internally and sends it to the FSMGR.

**RFSLTSEN_MCTRL -** This signal is set to 1 if the RFSLTS_MCTRL is valid.

## SIGNALS TO OMST

**DATA_MCTRL -** This 32 bit signal are the cell data from the CSTR. The MCTRL does the parallel to serial conversion internally.

# 7. OUTPUT MASTER (OMST)

## SIGNALS FROM MEMORY CONTROLLER

**DATA_MCTRL -** This 32 bit signal are the departing cell data from the CSTR.

## SIGNALS TO NETWORK INTERFACE

**TXPRTY -** The transmit parity signal indicates the parity of the TDATA.

**TDATA -** The transmit cell data bus carries the ATM cell. TDATA[15] is the MST, TDATA[0] is the LSB of the 16 bit data path.

**TWREN -** The channel active low transmit write enable inputs is used to initiate writes to the channel transmit FIFO.

**TSOC -** Start of cell signal.

**TCA -** Transmit cell available signal indicates when space for a cell is available in the channel transmit FIFO.

# APPENDIX B. OPERATIONAL SCENARIOS

The following diagrams show the status at the end of Cell Time 6 to Cell Time 11 in the operational examples in chapter 3.

# CELL TIME 6

**Queue Lookup Table**
(Tag, Queue ID)

**Queue Selector**

| | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

**Free Queue List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | - | 2 | 3 | 4 | 5 | 6 | 7 |

**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | D,15 | - | - | - | - | - | - | - | - | - | - | B,13 | E,16 | - | F,17 |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Free Slot List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

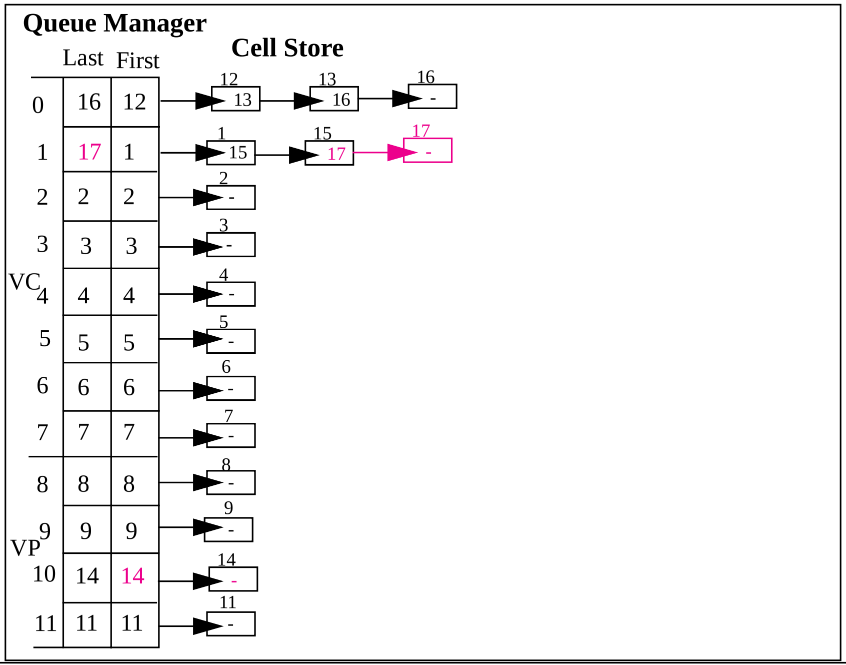
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

**Output Master**

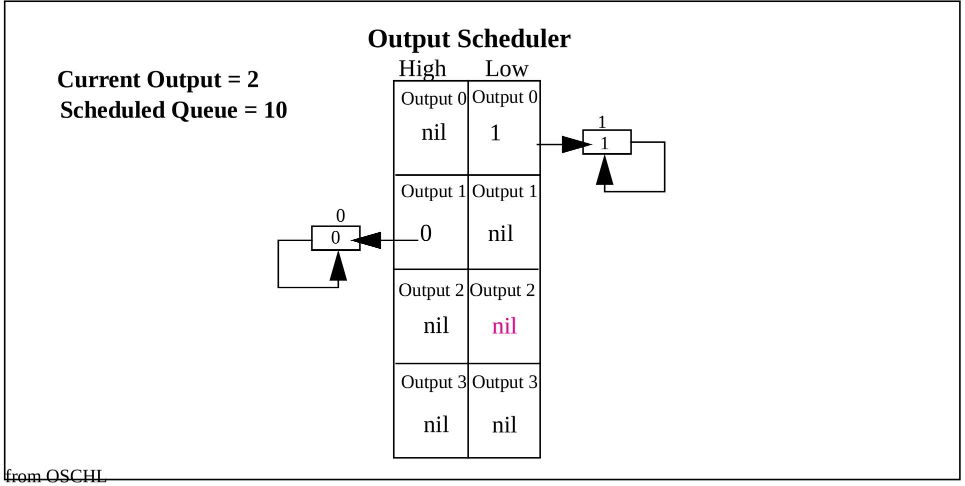
| | | | | | |
|---|---|---|---|---|---|
| - | - | - | - | - | Output 0 |
| - | - | - | - | A | Output 1 |
| - | - | - | - | C | Output 2 |
| - | - | - | - | - | Output 3 |

* Cell G, VP, Queue ID 10 * Cell F is written into Slot 15

* Queue 10 is selected, Cell C is sent to OMST, return slot 10, remove queue 10 from OSCHL



| | Last | First |
|---|---|---|
| 0 | 16 | 12 |
| 1 | 17 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| VC 4 | 4 | 4 |
| 5 | 5 | 5 |
| 6 | 6 | 6 |
| 7 | 7 | 7 |
| 8 | 8 | 8 |
| VP 9 | 9 | 9 |
| 10 | 14 | 14 |
| 11 | 11 | 11 |



**Current Output = 2**
**Scheduled Queue = 10**

| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>1 |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>nil |
| Output 3<br>nil | Output 3<br>nil |

## CELL TIME 7

**Queue Lookup Table**
(Tag, Queue ID)

**Queue Selector**

| | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | (01,2) |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

**Free Queue List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | - | - | 3 | 4 | 5 | 6 | 7 |

**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | D,15 | - | - | - | - | - | - | - | - | - | - | B,13 | E,16 | G,18 | F,17 |
| **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

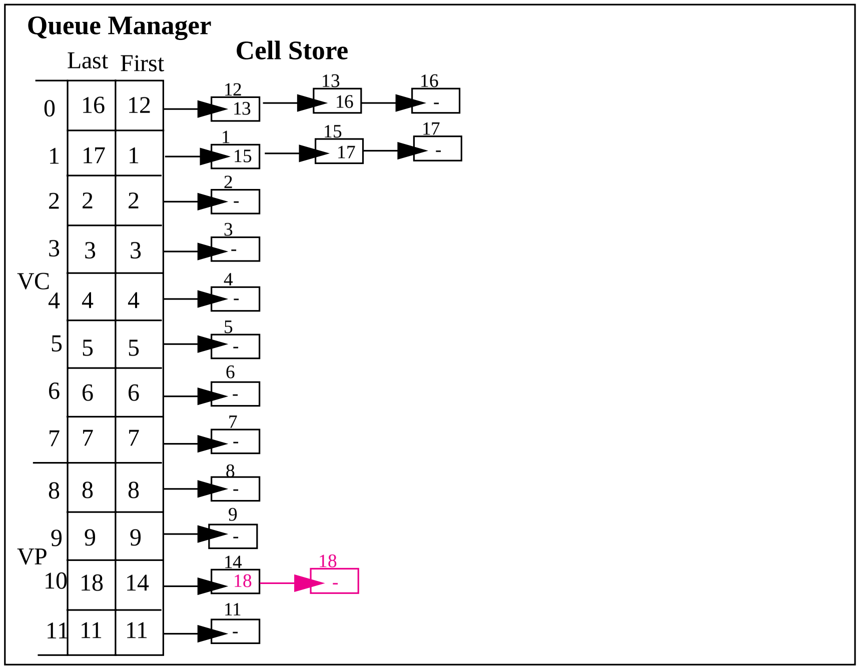
**Free Slot List**

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

**Output Master**

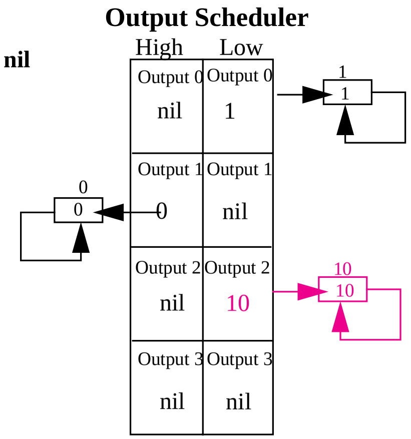
| | | | | | | |
|---|---|---|---|---|---|---|
| | - | - | - | - | - | Output 0 |
| | - | - | - | - | A | Output 1 |
| | - | - | - | - | C | Output 2 |
| | - | - | - | - | - | Output 3 |

* Cell H, VC, Index 2, Tag 01, Queue ID 2



| | | Last | First |
|---|---|---|---|
| | 0 | 16 | 12 |
| | 1 | 17 | 1 |
| | 2 | 2 | 2 |
| | 3 | 3 | 3 |
| VC | 4 | 4 | 4 |
| | 5 | 5 | 5 |
| | 6 | 6 | 6 |
| | 7 | 7 | 7 |
| | 8 | 8 | 8 |
| | 9 | 9 | 9 |
| VP | 10 | 18 | 14 |
| | 11 | 11 | 11 |

**Current Output = 3**
**Scheduled Queue = nil**



| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>1 |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>10 |
| Output 3<br>nil | Output 3<br>nil |

# CELL TIME 8

**Queue Lookup Table**
(Tag, Queue ID)

**Queue Selector**

| | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | (01,2) |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

**Free Queue List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| - | - | - | 3 | 4 | 5 | 6 | 7 |

**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | H,19 | - | - | - | - | - | - | - | - | - | B,13 | E,16 | G,18 | F,17 |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Free Slot List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

**Output Master**

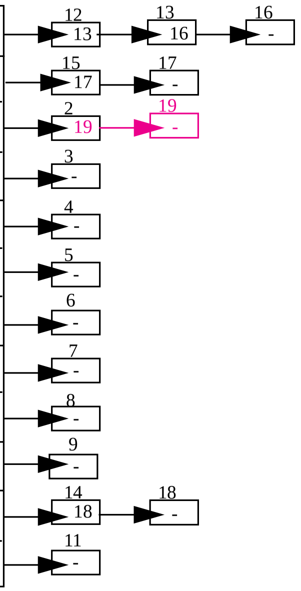
| | | | | | | |
|---|---|---|---|---|---|---|
| | - | - | - | - | D | Output 0 |
| | - | - | - | - | A | Output 1 |
| | - | - | - | - | C | Output 2 |
| | - | - | - | - | - | Output 3 |

* No new cell coming
* Queue 2 is added to Output 1, High Priority, Cell H is written into Slot 2
* Queue 1is selected, Cell D us sent to OMST, return slot 1

**Queue Manager**

| | | Last | First |
|---|---|---|---|
| VC | 0 | 16 | 12 |
| | 1 | 17 | 15 |
| | 2 | 19 | 2 |
| | 3 | 3 | 3 |
| | 4 | 4 | 4 |
| | 5 | 5 | 5 |
| | 6 | 6 | 6 |
| | 7 | 7 | 7 |
| VP | 8 | 8 | 8 |
| | 9 | 9 | 9 |
| | 10 | 18 | 14 |
| | 11 | 11 | 11 |

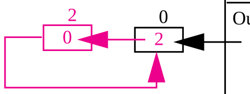
**Cell Store**

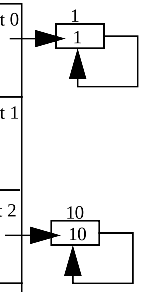


**Current Output = 0**
**Scheduled Queue = 1**

**Output Scheduler**

| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>1 |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>10 |
| Output 3<br>nil | Output 3<br>nil |





## CELL TIME 9

### Queue Lookup Table

(Tag, Queue ID)

| Queue Selector | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | (--, -) |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

### Free Queue List

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 2 | - | - | 3 | 4 | 5 | 6 | 7 |

### Cell Store

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | - | B,13 | E,16 | G,18 | F,17 |
| **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

### Free Slot List

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

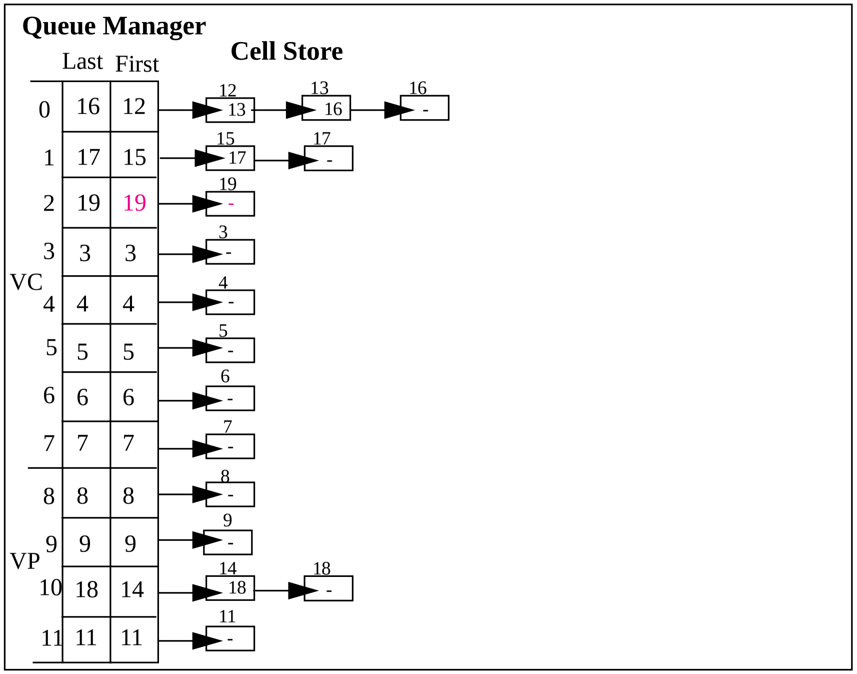
### Output Master

| | | | | | | |
|---|---|---|---|---|---|---|
| - | - | - | - | D | Output 0 |
| - | - | - | H | A | Output 1 |
| - | - | - | - | C | Output 2 |
| - | - | - | - | - | Output 3 |

* Cell I coming in, VP, Queue ID 11

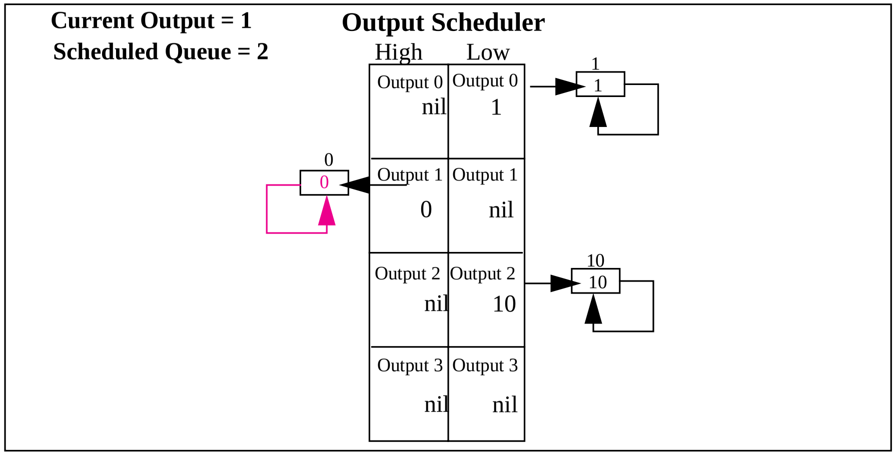
* Return Queue ID 2 to free queue list, invalidate the entry in lookup table

* Queue 2 is selected, Cell H is sent to OMST, return slot 2, Queue 2 is removed from OSCHL



### Queue Manager

| | | Last | First |
|---|---|---|---|
| VC | 0 | 16 | 12 |
| | 1 | 17 | 15 |
| | 2 | 19 | 19 |
| | 3 | 3 | 3 |
| | 4 | 4 | 4 |
| | 5 | 5 | 5 |
| | 6 | 6 | 6 |
| | 7 | 7 | 7 |
| VP | 8 | 8 | 8 |
| | 9 | 9 | 9 |
| | 10 | 18 | 14 |
| | 11 | 11 | 11 |



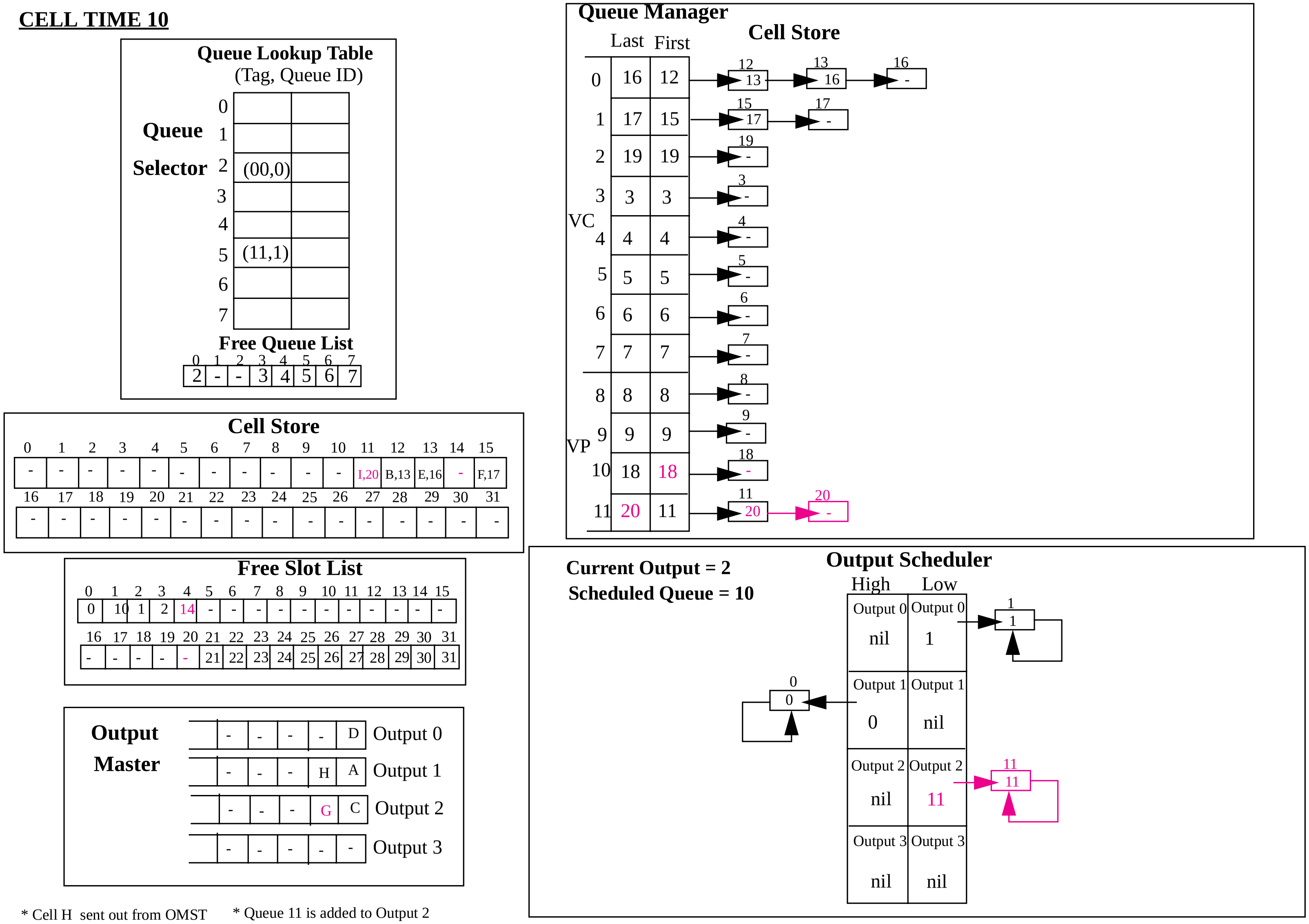
Current Output = 1

Scheduled Queue = 2

### Output Scheduler

| | High | Low |
|---|---|---|
| | Output 0<br>nil | Output 0<br>1 |
| | Output 1<br>0 | Output 1<br>nil |
| | Output 2<br>nil | Output 2<br>10 |
| | Output 3<br>nil | Output 3<br>nil |

## CELL TIME 10



**Queue Lookup Table**
(Tag, Queue ID)

| Queue Selector | | |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | (00,0) | |
| 3 | | |
| 4 | | |
| 5 | (11,1) | |
| 6 | | |
| 7 | | |

**Free Queue List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 2 | - | - | 3 | 4 | 5 | 6 | 7 |

**Cell Store**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | I,20 | B,13 | E,16 | - | F,17 |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Free Slot List**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10 | 1 | 2 | 14 | - | - | - | - | - | - | - | - | - | - | - |

| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

**Output Master**

| | | | | | | |
|---|---|---|---|---|---|---|
| | - | - | - | - | D | Output 0 |
| | - | - | - | H | A | Output 1 |
| | - | - | - | G | C | Output 2 |
| | - | - | - | - | - | Output 3 |

**Queue Manager**

| | | Last | First |
|---|---|---|---|
| VC | 0 | 16 | 12 |
| | 1 | 17 | 15 |
| | 2 | 19 | 19 |
| | 3 | 3 | 3 |
| | 4 | 4 | 4 |
| | 5 | 5 | 5 |
| | 6 | 6 | 6 |
| | 7 | 7 | 7 |
| VP | 8 | 8 | 8 |
| | 9 | 9 | 9 |
| | 10 | 18 | 18 |
| | 11 | 20 | 11 |

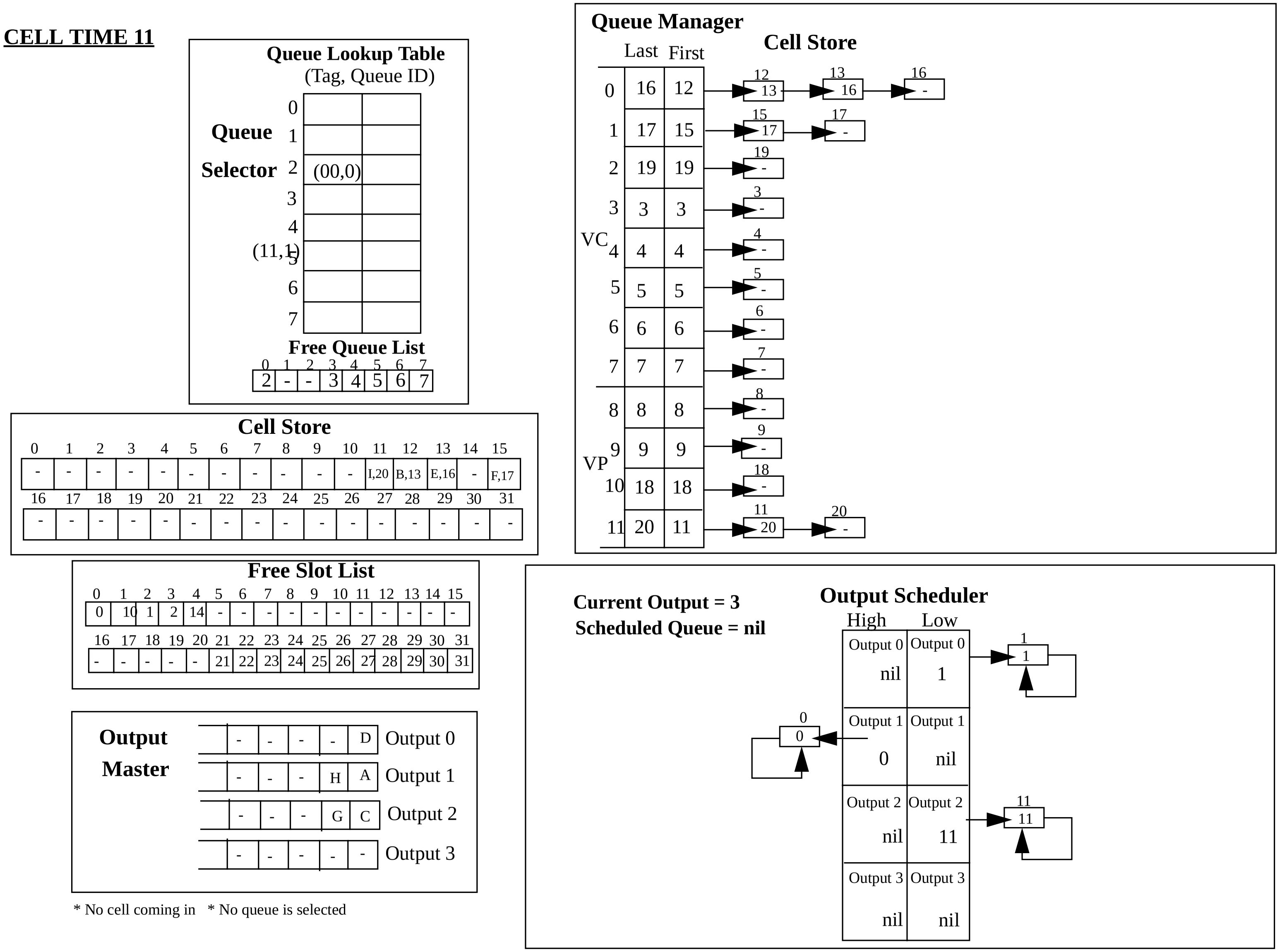
**Current Output = 2**
**Scheduled Queue = 10**

**Output Scheduler**

| High | Low |
|---|---|
| Output 0<br>nil | Output 0<br>1 |
| Output 1<br>0 | Output 1<br>nil |
| Output 2<br>nil | Output 2<br>11 |
| Output 3<br>nil | Output 3<br>nil |

* Cell H  sent out from OMST     * Queue 11 is added to Output 2
* Queue 10 is selected, Cell G  to OMST, return slot 14, Queue 10  is deleted from OSCHL

## CELL TIME 11



* No cell coming in * No queue is selected

# REFERENCES

[1] The ATM Forum Technical Committee, “UTOPIA Level 2, v1.0”, af-phy-0039.000.

[2] J. C. R. Bennett, D. C. Stephens and H. Zhang, “High Speed, Scalable, and Accurate Implementation of Packet Fair Queueing Algorithms in ATM Networks,” Proceedings of IEEE ICNP’97, oct., 1997, 7-14.

[3] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, J. S. Turner, “Design of a Gigabit ATM Switch”, Infocom 1997.

[4] Maurizio Casoni, J.S. Turner, “Improved Analysis of Early Packet Discard,” Proceedings of the International Teletraffic Congress, June, 1997.

[5] H. Jonathan Chao, “An ATM Queue Manager Handling Multiple Delay and Loss Priorities,” IEEE/ACM Tran. on Networking, vol. 3, no. 6, Dec. 1995, pp 652-659.

[6] Yuhua Chen, J. S. Turner, “Dynamic Queue Assignment in a VC Queue Manager for Gigabit ATM Networks,” Proceedings of IEEE ATM Workshop, 1998

[7] Yuhua Chen, J. S. Turner, “Design of a Weighted Fair Queueing Cell Scheduler for ATM Networks,” submitted to IEEE GLOBECOM, 1998.

[8] R. Chipalkatti, J. F. Kurose, and D. Towsley, “Scheduling Policies for Real-Time and Non-real-Time Traffic in a Statistical Multiplexer,” Proc. 1989 GLOBECOM conf., 1989, pp 774-783.

[9] Haoran Duan, J. W. Lockwood, et al., “A High-Performance OC-12/OC-48 Queue Design Prototype for Input -Buffered ATM Switches,” INFOCOM’97.

[10] Massoud R. Hashemi, Alberto Leon-Garcia, “A General Purpose Cell Sequencer/ Scheduler for ATM Switches,” INFOCOM’97.

[11] Manolis Katevenis, Stefanos Sidiropoulos, and Costas Courcoubetis, “Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip,” IEEE Journal on Selected Areas in Communication, vol. 9, No. 8, Oct. 1991, pp. 1265-1279.

[12] Paul Landsberg, Charles Zukowski, “Generic Queue Scheduling: Concepts and VLSI,” INFOCOM’94, pp. 1438-1445.

[13] A. A. Lazar, A. Temple, and R. Gidron, “A Metropolitan Area Network Based on Asynchronous Time Sharing,” IEEE ICC, June 1989, pp. 630-634.

[14] Duan-Shin Lee,”Generalized Longest Queue First: An Adaptive Scheduling Discipline for ATM Networks,” INFOCOM’97.

[15] Yoshihiro Ohba, “QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM networks,” INFOCOM’97.

[16] Allyn Romanow, Sally Floyd, “Dynamics of TCP Traffic over ATM Networks,” IEEE Journal on Selected Areas in Communications, vol. 13, no. 4, May 1995, pp. 633-641.

[17] Ion Stoica, Hui Zhang, T. S. Eugene Ng, A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service,” SIGCOMM’97.

[18] J. S. Turner, “Maintaining High Throughput During Overload in ATM Switches,” INFOCOM’96, pp. 287-295.

[19] Dallas E. Wrege, Jorg Liebeherr, “A Near-Optimal Packet Scheduler for QoS Networks,” INFOCOM’97.