FORTH-ICS / TR-172 July 1996

The Architecture, Operation and Design of the Queue Management Block in the ATLAS I ATM Switch

Christoforos E. Kozyrakis

Among the various switch buffer architectures, output queueing implemented in a completely shared buffer is the one that achieves the highest possible utilization of both output bandwidth and buffer space. The high link throughput, small cell size and additional features of ATM switching, such as multiple classes of service, multicasting and flow control, enforce further extensions to the above scheme and demand pure hardware implementations. In this work we present the hardware block maintaining output queues per priority class in the ATLAS I single chip ATM switch. It also provides support for multicasting and multi-lane credit-based flow control. Techniques such as pipelined and superscalar processing, usually employed in processors' design, are used in order to accommodate for the amount and high speed of operation required. This also modifies the approach to the timing of operations, the control design and the calculation of the hardware complexity. The block was extensively simulated to ensure the correctness of its operation. Although the hardware implementation is currently in progress, the circuits already laid out are presented, while the VLSI design of the remaining blocks is analyzed. In addition, the Priority Enforcer circuit and its full-custom layout is thoroughly described.

The Architecture, Operation and Design of the Queue Management Block in the ATLAS I ATM Switch

Christoforos E. Kozyrakis†

Institute of Computer Science (ICS)
Foundation for Research and Technology Hellas (FORTH)
Science and Technology Park, Heraklion, Crete
P.O. Box 1385, GR-711-10 Greece
email: koziraki@ics.forth.gr

Technical Report FORTH-ICS/TR-172 July 1996

Work Performed under ACTS Project 060 "ASICCOM" and as B.Sc. Thesis at the Univ. of Crete

ABSTRACT: Among the various switch buffer architectures, output queueing implemented in a completely shared buffer is the one that achieves the highest possible utilization of both output bandwidth and buffer space. The high link throughput, small cell size and additional features of ATM switching, such as multiple classes of service, multicasting and flow control, enforce further extensions to the above scheme and demand pure hardware implementations. In this work we present the hardware block maintaining output queues per priority class in the ATLAS I single chip ATM switch. It also provides support for multicasting and multi-lane credit-based flow control. Techniques such as pipelined and superscalar processing, usually employed in processors' design, are used in order to accommodate for the amount and high speed of operation required. This also modifies the approach to the timing of operations, the control design and the calculation of the hardware complexity. The block was extensively simulated to ensure the correctness of its operation. Although the hardware implementation is currently in progress, the circuits already laid out are presented, while the VLSI design of the remaining blocks is analyzed. In addition, the Priority Enforcer circuit and its full-custom layout is thoroughly described.

KEYWORDS: VLSI switches, ATM switches, ATLAS I switch, shared buffer, credit-based flow-control, multiple output queues, queue management, pipelining, priority enforcer.

This text is available in Postscript form, by anonymous ftp, from server "ftp.ics.forth.gr".
Dirctory: "tech-reports/1996"
Files: "1996.TR172.QueueMangement.README", "1996.TR172.QueueMangement.ps.gz"

† The author is also with the Computer Science Department, University of Crete, Greece.

Contents

1. Introduction

Asynchronous Transfer Mode (ATM) [LeBo92] puts additional requirements both on the speed and the complexity of switches (routers), used as building blocks in networks. The main reasons for that are : a) the high cell arrival and departure rates (up to millions of cells per second), due to the high bandwidth of the links and the small cell size; b) the small delay that cells are expected to undergo through the switch; c) the high utilization of the output throughput demanded; c) the fact that, since ATM classifies network traffic according to the quality of service requirements that has to guarantee, switches must route cells in a priority-based manner; and e) other features desirable in high-speed networks such as multicasting and flow control. In order to meet these demands, switches must use flexible buffer architectures and implement high performance data structures for cells stored in them, which were not essential before [CoST88]. Since these structures must be updated in rates similar to those of cell arrivals and departures, they must be implemented in hardware [Toba90].

Output queues, implemented as linked lists in a completely shared buffer, have been identified as the combination of data structure and buffer architecture that results in high utilization of both available throughput and buffer space [HlKa88][TaFr88]. This organization can be used in ATM switches after properly extending it to support multiple classes of service, multicasting and flow-control. In this thesis, we present the *Queue Management* block of the *ATLAS* I ATM switch, that maintains queues per output and per service class, along with queues for multicast cells and cells blocked in the switch by the flow control protocol.

1.1 The ATLAS I switch

ATLAS I [KaSV96][KSVMC96] is a single-chip ATM switch currently developed at ICS-FORTH within the ASICCOM [1] project. Its intended use is as building block in high-throughput and low-latency networks, varying from local area (LAN) to wide area (WAN) and desktop area (DAN) networks.

ATLAS has 16 input and 16 output point-to-point links, each running at 622 Mbits/s. Its aggregate throughput reaches 20 Gigabits/second. It provides shared buffer for 256 ATM cells, using the pipeline memory architecture [KaVE95]. *ATLAS* also supports configurable VP and VP-VC switching (by using a translation table), both rate and optional credit-based multi-lane back-pressure flow control [KaSS96], load monitoring, link bundling and merging of flow groups. Multicasting is also supported, as long as all the copies of the cell transmitted to different links use the same VP/VC identifier.

Internally, the switch has an additional input and output. Thus, it functions as a 17x17 switch. The 17th input is used for inserting cells in the switch from the Switch Control and Monitoring block, while

[1] The ASSICOM project is part of the European Union ACTS (Advanced Communication Technologies and Services) Programme.

the 17th output delivers outgoing cells to this block. These two ports enable the switch management and control through cell transmissions without interfering with the normal operation.

In order to provide the mechanisms for support of various quality of service requirements, the *ATLAS* I switch recognizes three classes of cells, differentiated by their priority level. While the high priority class is non back-pressured, since it is intended for real-time traffic, the medium and low priority data are flow-controlled and intended for VBR-ABR and UBR data respectively. Switch resources, e.g. buffer space, can be partially reserved for each class by using various cell counters in order to define buffer partitions and other limits. The priority of each incoming cell is specified in the corresponding entry for its flow group (VP or VP/VC) in the translation table.

The operation of the switch is pipelined and can accommodate for back-to-back and parallel arrivals and departures of cells, as well as for the execution of the credit-based flow control protocol, through reception and transmission of credits. The switch will be fabricated in a 0.5μm CMOS technology and its clock frequency will be 50 MHz.

1.2 The Queue Management Block

The *Queue Management* [2] block is the part of the *ATLAS* I switch responsible for implementing the appropriate data structures for cells within its shared data buffer [KSVMC96]. These structures are used in order to keep record of both cells blocked by the flow control protocol and cells ready to be transmitted to their destination, and to be able to serve them in the way defined by the flow control scheme and the priority rules.

Cells blocked by the credit-based flow control protocol, i.e. cells without all the credits corresponding to their flow group and their destination links, are kept in the CreditLess Cell List (CLL). This list is implemented as a pool of cells, without any special connectivity. A cell in the data buffer (DB) belongs to the CLL, when its routing information has been written in the corresponding memories of the block, and it is not included in any other structure. Cells remain in the CLL until they receive credits for all their destination links.

Cells ready to be leave the switch are inserted in ready queues. Ready queues are FIFO structures implemented as linked lists by using head, tail and next cell pointers. There are 54 such queues maintained by the *QM* block : one per output and per priority level for unicast cells (16x3), three for cells destined to the Switch Control and Monitoring block (17th output), plus three queues for the multicast cells (one for each priority class). Maintaining queues per output enables the switch to fully utilize output throughput, while queues per priority level make priority-based routing possible. Multicast cells are placed in separate queues so that links are properly reserved for their transmission when possible, without unnecessarily delaying any unicast cells. Naturally, the best solution would be to enqueue each

[2] The *Queue Management* block will be frequently referred as the *QM* block in this document.

multicast cell in every unicast ready queue corresponding to a link it must be transmitted to. Yet, this would require extra memory space for next cell pointers, since each cell in the data buffer could be in up to 16 ready queues at the same time. In order to avoid sacrificing that much memory to pointers, separate multicast queues were preferred.

The *Queue Management* block operates on cells and credits in a pipelined and parallel manner, as explained later. Operations on incoming cells are performed by using their header and the information attached to it after passing through the VP/VC Routing and Translation Table. In a similar way, departing cells are served by producing their new header and their address in the data buffer, and forwarding them to the proper outlink circuits. The whole credit, as read from the Credit Extraction and Serialization block, is used for credit operations.

1.3 This thesis

In this work, we examine in details the *Queue Management* block, the parts it consists of, its operation and its complexity.

Section 2 presents the organization of the block and the memories it contains. In section 3 we thoroughly present its operation, while section 5 investigates the number of ports per memory needed for these actions. Section 6 describes the block control logic, and section 7 explains data hazards due to the pipelined operation of the block and the way they are handled. In section 8, the simulation methods used are presented. Section 9 describes the full-custom layout of the two-ported memory blocks and the the VLSI implementation of the remaining circuits. Section 10, presents the operation of the Prioriry Enforcer circuit, the design alternatives, the implementation and its full-custom layout for the *QM* block. Finally, section 11 describes conclusions and future work.

2. Block Organization

The *Queue Management* block mainly consists of five memories. One of them, the Head-Tail register file, holds the necessary head and tail pointers for maintaining the ready queues, while the remaining four hold routing information and characteristics of the cells stored in the switch. The latter correspond each of their entries to a slot in the data buffer and keep there all information about the cell occupying that slot. The five memory blocks and their purpose are :

VPout Memory : it holds the new routing information for each cell in the data buffer. This is a 12-bit field (VP/VCout) used to replace an equal-sized field in the VP/VC section in the cell header. This field is read from the Routing and Translation Table of the switch on cell arrival. Size: 256 x 12 bits.

OutMask Memory : it contains a mask, indicating the link(s) that the corresponding cell in the data buffer should be transmitted to (16 bits, one bit per outlink). This mask, called outmask, is also read from the Routing Table on cell arrival. Although the Routing Table provides a 17th bit as well, identifying cells destined to the 17th outport (Switch Control and Monitoring block), this bit is not stored in the OutMask memory. Size: 256 x 16 bits.

CreditMask Memory : it is used only with cells of flow-controlled groups and stores a mask identifying currently available credits to each cell in the switch (called creditmask). This mask is 16 bits long, since there can be up to one credit available per outlink per flow-group. Its original contents are read from the Credit Table memory on cell arrival, and are properly updated, each time a credit for that flow-group arrives. An additional (17th) bit is used to discriminate between medium and low priority flow- controlled cells. Size: 256 x 17 bits.

LinkList Memory : it holds a pointer to the next cell in the ready queue to which the corresponding cell in the DB belongs (if it does). This is an 8-bit address to the data buffer. It also contains the primary outlink that the next cell must be transmitted to, encoded in 4 bits. This is necessary for properly reserving links for the transmission of multicast cells (explained in details later). Size: 256 x 12 bits.

Head-Tail register file (HTRF) : this block has 54 entries where it holds the head and tail pointers of each ready queue. All pointers are 8 bits long and index to a cell in the data buffer. There is a redundant bit per entry, indicating whether the queue is empty or not (Valid bit), which is used to accelerate the operation of the block. Size: 54 x 17 bits.

The VPout, OutMask and CreditMask memories comprise the CreditLess Cell List sublock, while the LinkList memory with its peripheral circuits form the Ready Queues sublock [KSVMC96].

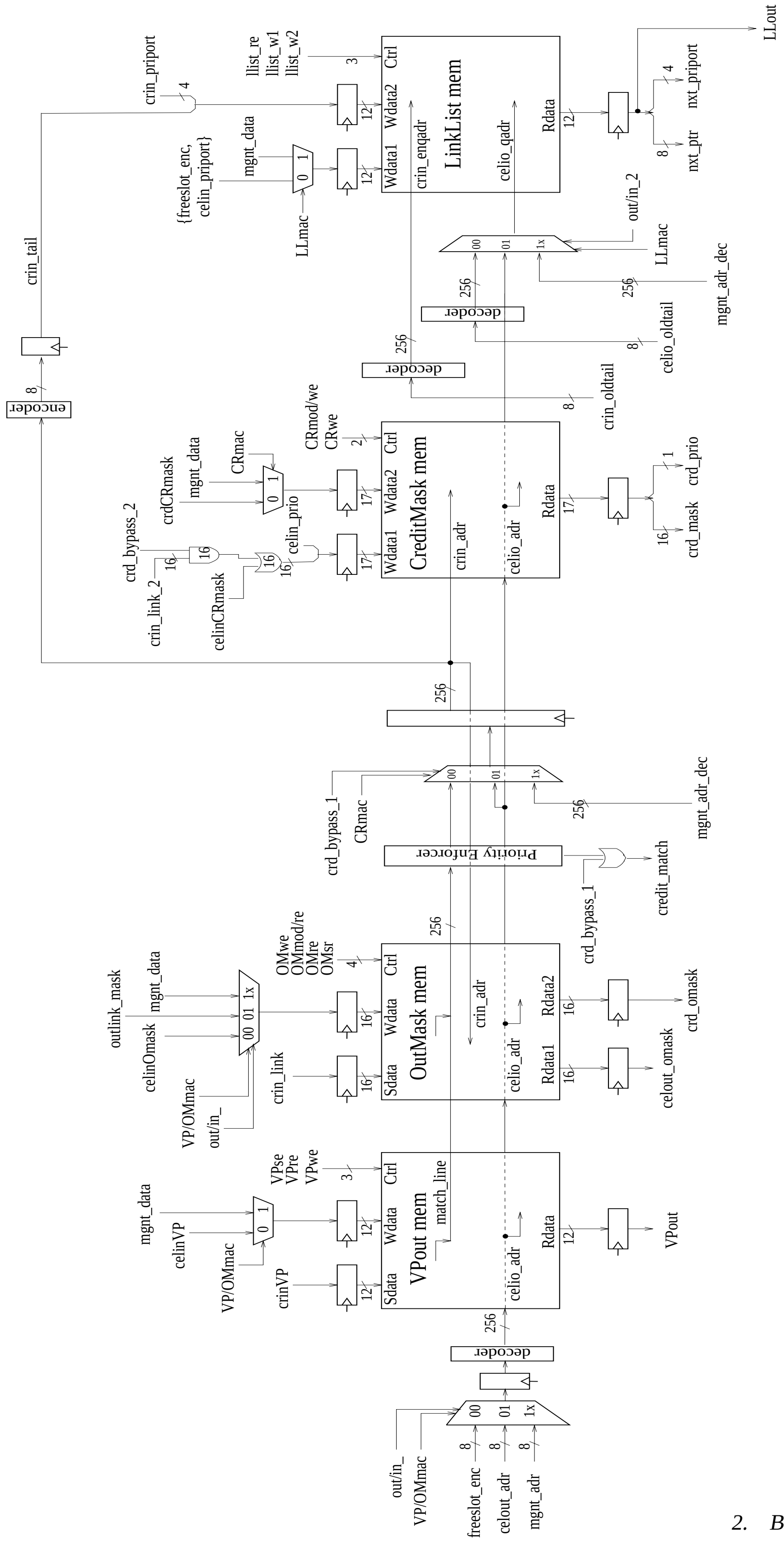


Figure 1: The *Queue Management* block main datapth.

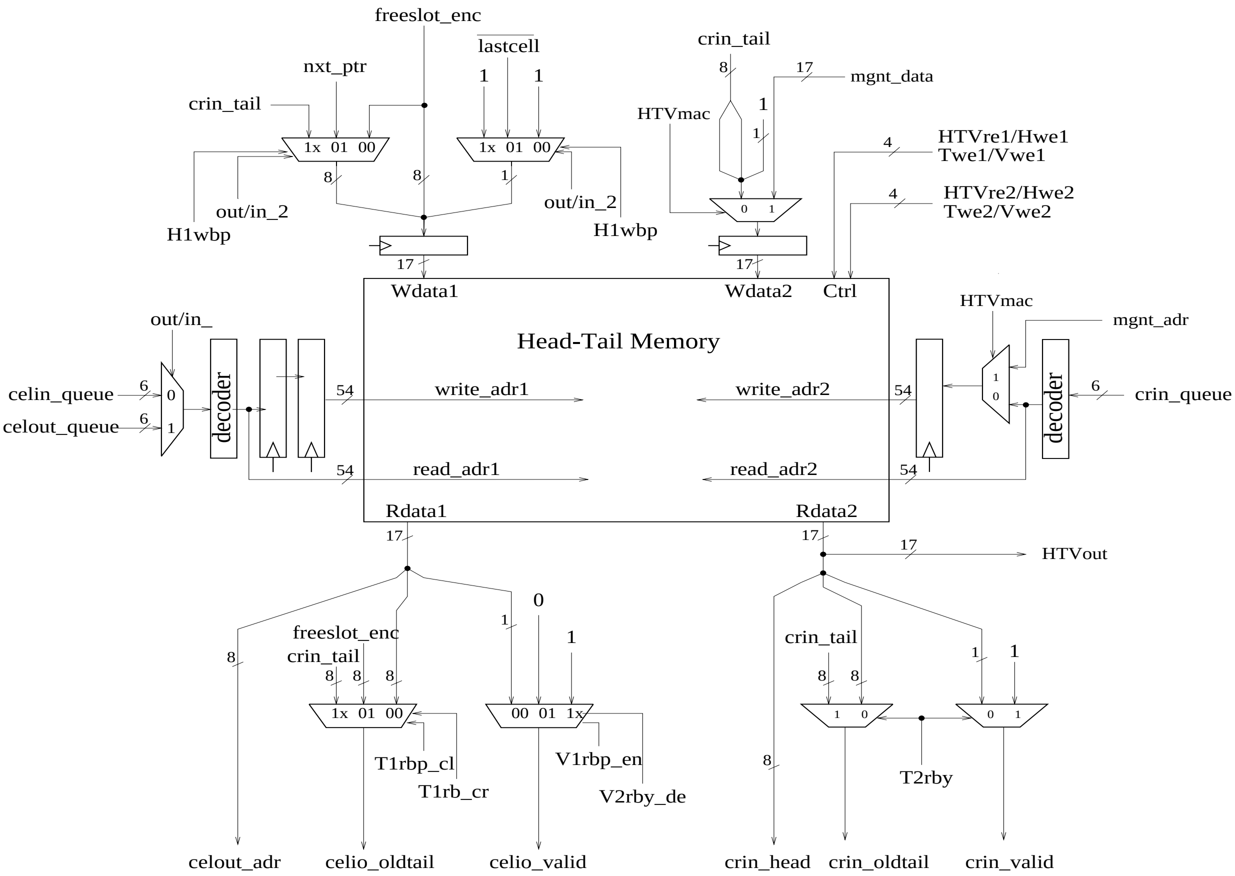


Figure 2: The Head-Tail Pointer register file and its peripheral circuits.

Another important part of the block is the control logic. Since comprehension of block operation is required in order to understand the organization of control logic, its description is presented later.

The block also contains a number of cells such as registers, multiplexors, decoders and other combinational circuits and gates, necessary for correct operation. Figure 1 presents the complete diagram of the main datapath of the block, while figure 2 is the block diagram of the Head-Tail register file and its peripheral circuits. The purpose and the operation of certain parts of these diagrams will become clear in the rest of this document.

3. Block Operation

The purpose of the *Queue Management* block is to properly maintain the ready queues and the creditless list on each event. There are three types of events in the *ATLAS* I switch that effect these queues and have to be properly handled. These events are cell arrivals, cell departures and credit arrivals.

All cell and credit events are served by the *QM* block, and the switch in general, in a pipelined fashion [KSVMC96]. One can imagine the block as a two way superscalar pipelined CPU, where the first processing unit is used for credit arrival operations, while the second one is shared by cell arrivals and departures. Since pipelined and superscalar processing is employed, actions related to different events can be simultaneously in progress. Following, there is detailed description of the pipeline stages, the actions and accesses performed on each event, based on the diagrams in figures 1 and 2 and the names of signals and blocks declared there. In addition, table 1 summarizes the main actions per event and per pipeline stage.

3.1 Cell Arrival Operation

Incoming cells start being processed by the *QM* block as soon as the Scheduler block decides to serve them. This is whenever there is a clock cycle when no cell transmission to a link can be initiated. The necessary actions are performed within three pipeline stages (3 clock cycles), as shown in table 1.

In the first cycle, an empty slot for the cell to be stored in is requested from the Free List block, which keeps track of all empty slots in the shared data buffer. Its address is given through the freeslot_enc pointer. The corresponding slots of the *QM* memories are used for holding information about this cell. In parallel to that, the block holding the cell counters is accessed to find out if the cell can be accepted by the switch or not, i.e. whether it will cause an overflow of the corresponding buffer partition or not. In case all slots in the DB are occupied or the cell is dropped, the actions of the following stages are not performed.

In the same time, the VP/VCout information read from the Routing and Translation Table, is used as an address to read available credits on all links for the flow group to which the cell belongs, from the Credit Table memory. The cell "consumes" all credits available for its destination links (or outlinks) by clearing the corresponding bits in the Credit Table entry, while setting them in its creditmask (celinCRmask). Cells of non flow-controlled groups are always presented with credits for all their outlinks. The same holds for cells coming from the 17th input (Switch Control and Monitoring block), which are always unicast cells. This information is also used to decide whether this cell is ready [3] and should be enqueued in a ready queue, or it should be added to the CLL. Finally, the class identifier and

[3] A cell is considered ready when it belongs to a non flow-controlled group, or there are credits available for all its destination links.

the 16 least significant bits of the outlink mask of the cell, read from the Routing Table as well, are used to form a HTRF address (celin_queue) in order to read the tail pointer (celio_oldtail) and the valid bit (celio_valid) of the ready queue to which the cell corresponds.

During the next clock cycle, the VP/VCout field (celinVP), the outlink mask (celinOmask) and the creditmask (celinCRmask) of the cell are stored in the corresponding memories, in the slots indexed by its DB address (celio_adr). If the cell is ready and its corresponding queue is not empty at the time (valid bit set), its address (freeslot_enc) and primary outlink (celin_priport) are written in the LinkList entry indicated by the tail pointer (celio_qadr). In this way, the cell is added at the tail of the ready queue. The accesses of the next stage are performed only for ready cells.

In the last cycle, the DB address of the ready cell (freeslot_enc) is written at the tail pointer entry of the queue slot in the HTRF, indexed by write_adr1 vector. If the queue was previously empty (valid bit reset), the head pointer entry and the valid bit are updated as well.

Event	Stage (cycle) 1	Stage (cycle) 2	Stage (cycle) 3	Stage (cycle) 4
Cell Arrival	Search Free List Read Cell Counters Read Tail Ptr Read Credit Table	Write VP/VCout Write OutMask Write CreditMask Write Next Ptr	Update Tail Ptr	
Cell Departure	Read Head Ptr	Read VP/VCout Update/Read OutMask Read Next Ptr	Update Head Ptr	
Credit Arrival	Search VP/VCout Search OutMask	Update/Read CreditMask Read OutMask	Read Tail Ptr	Write Tail Ptr Write Next Ptr

Table 1: The three event types and the main actions per pipeline stage of their operations.

3.2 Cell Departure Operation

The departure of a cell from a ready queue to a specific outlink is also initiated by the Scheduler block. This is done as soon as the link is idle (or about to become idle) and the priority rules allow that. The proper actions are performed within three pipeline stages (3 clock cycles).

In the first cycle, the head and tail pointers of the ready queue from which the cell departs, are read from the HTRF. The address of the queue (celout_queue) is calculated by the Scheduler block, using the service class of the cell and the identity of the outlink. The head pointer (celout_adr) identifies the cell to be transmitted and is used as an address in all memory accesses during the following cycle, while the tail pointer (celio_oldtail) is only needed to detect if the cell is the last one in the queue. This is the case where the head and the tail pointers are equal.

During the second cycle, the VP/VCout field (VPout) of the cell is read and forwarded to the proper outlink circuits in order to form the new cell header. At the same time, the pointer to the next cell in the queue (nxt_ptr) and its primary outlink (nxt_priport) are read from the LinkList memory. Finally, the bit of the cell OutMask entry, corresponding to the selected outlink, is reset. This bit is indicated in the outlink_mask vector. The rest 15 bits in the entry are read from the OutMask memory (celout_omask field). If all these bits are 0, the cell has just been transmitted to its last destination and should be dequeued. In this case, the Free List block and the block keeping the various cell counters are notified in order to mark the slot free and decrement or increment the proper counters. This access is a useless read in the case of a cell destined to the 17th output, since both the outlink_mask and the OutMask memory entry will be zero masks.

The actions of the last cycle take place only when the cell must be dequeued. If the cell is not the last one in the queue, the head pointer entry in the Head-Tail register file is updated with the address of the next cell (nxt_ptr). Otherwise, the valid bit of the queue is reset to indicate that the queue is now empty. The address of the slot corresponding to the queue served, is indexed by the write_adr1 vector.

3.3 Credit Arrival Operation

Serving an incoming credit is initiated by the Credit Serialization block, whenever an unprocessed credit exists. The necessary actions take up 4 pipeline stages (4 clock cycles).

During the first cycle, the flow-group identifier of the credit (crinVP) and a 16-bit mask indicating the arrival link (crin_link) are used to search the VPout and OutMask memories respectively, for cells in the CLL waiting for that credit. Both masks are available from the Credit Serialization block. While we demand a complete match of an entry with the search pattern in the VPout memory, the search access in the OutMask memory aims to detect entries with the corresponding outlink enabled (not complete matching of masks). Thus, logic zeros in the search mask are treated as don't-care values. Because of the merging of flow-groups supported by the *ATLAS* I switch, there can be more than one entries matching during this search, i.e. more than one cells expecting this credit. Hence, a priority enforcer must operate on the result of the action (match_line vector), in order to select the cell to which the credit will be handed. It also generates the credit_match signal, indicating whether the search operation was successful or not. The priority enforcer will probably be a cyclic one, in order to maintain some random selection properties.

If no cell was found during the search, the credit arrival is noted in the Credit Table memory during the second cycle, by setting the bit corresponding to the arrival link in the proper entry. Otherwise, the output of the priority enforcer, indicating the decoded address of the selected cell (crin_adr), is used to update its creditmask. The bit corresponding to the credit arrival link, identified by the crdCRmask vector, is set, while all the rest are read (crd_omask), along with the bit indicating the cell priority level (crd_prio). At the same time, the outmask of the cell (crd_omask) is read from the OutMask memory and compared to its creditmask in order to decide if the cell is ready and, therefore, must be enqueued. Actions in the following cycles take place only for ready cells.

In the next cycle, the tail pointer (crin_oldtail) and the valid bit (crin_valid) of the queue to which the cell corresponds, are read from the HTRF. The address of the queue (crin_queue) is calculated from the cell's outmask (crd_outmask) and the 17th bit of its creditmask (crd_prio), that indicates its priority level (medium or low). This information was read in the previous cycle.

During the last cycle, the cell is actually enqueued. If the ready queue was previously not empty, the address of the cell in the DB (crin_tail) and its primary outlink (crin_priport) are stored in the LinkList entry indexed by the tail pointer (crin_enqadr). In parallel, the same address is stored in the tail pointer entry (using the write_adr2 vector as an address). If the queue was empty before, no writing in the LinkList memory takes places and, apart from the tail pointer, the head pointer and the valid bit of the queue are properly set as well.

3.4 Additional Operations

In addition to the operations mentioned above, the *QM* block has an extra task to perform. It must provide the Scheduler block with information about the status of the ready queues. Based on this information, the Scheduler can decide how to serve ready cells as soon as possible, how to maintain the proper priority rules among them and how to initiate serving incoming cells, when no cell transmission is feasible. This is accomplished by maintaining three 17-bit amd three 16-bit masks. The first three masks (not_empty_high, not_empty_medium and not_empty_low) indicate whether the 51 (48 plus 3 for the 17th output) unicast ready cell queues are empty or not. The other three masks (reserve_high, reserve_medium and reserve_low) are used for the three multicast ready queues. Each mask indicates the outlinks (1 to 16) to which the cell currently at the head of the corresponding multicast queue must be transmitted to. Since all cells destined to the 17th outport are unicast, there is no need for a 17th bit in these masks. The Scheduler must reserve the links for the transmission of the multicast cell, as soon as priority rules allow that.

The bits of the not_empty masks are duplicates of the valid bits of the ready queues and are properly updated every they are modified. Reserve masks are updated every time there is a new cell at the head of a multicast queue. If the head is updated during a cell or credit arrival (enqueue in an empty queue), the outmask of the cell (celin_Omask or crd_omask respectively) is copied to the corresponding reserve mask, hence all the links are reserved at once. On the other hand, if the head is updated during a cell departure, the outmask of the new head is not available. In order to save an additional access to the OutMask memory, which would also raise the port requirements for that memory, the primary port field has been added to each LinkList entry. This field (nxt_priport) is used to set one bit in the proper reserve mask, when the address of the new head (nxt_priport) is read from the LinkList memory. When the cell is transmitted to this outlink, its whole outmask is read (celout_omask vector), and the rest of the bits in the reserve mask, those corresponding to the rest of the destination links, are set as well. Resetting the bits of reserve masks is performed by the Scheduler block.

3.5 Further details on block operation

There are certain actions described above, that do not serve any obvious reason or it is difficult to understand their necessity. In the following paragraphs, we present these actions and explain their purpose.

First of all, it is clear that the valid bit added to each entry in the HTRF is redundant. The information it provides, whether a ready queue is empty or not, can also be extracted from the not_empty and reserve masks. Yet, this bit was added because of the complexity of the circuit needed to extract this information for a specific queue from these masks. Maintaining the valid bit has no other cost than the memory bits it consumes and demands no extra ports for the HTRF, since it is written and read at the same time with the head and tail pointers. In addition to that, getting the queue's status from the valid bit and not from the masks simplifies the bypass control datapaths as explained later.

The OutMask memory, that holds information about the outlinks that each cell must be transmitted to, does not have a 17th bit which would indicate that a cell must be delivered to the Switch Control and Management block (17th output). This causes no problem, because all cells destined to the 17th output are unicast and are always presented with the proper credit. Thus, they are immediately added in the proper ready queue and no incoming credit has to be handed to one of them. Since the 17th bit in each OutMask entry would just be set on cell arrival reset on cell departure, we decided to remove it.

One can also notice that, updating the tail pointer when enqueueing an incoming cell could be performed one clock cycle earlier, in the second pipeline stage. All essential information in order to decide if this action is necessary and what should be written in the register file is available at the end of the first stage. What is more, setting the tail pointer as soon as possible, reduces the number of data hazards and bypass datapaths. Yet, we prefer to delay this write access for one clock cycle, so that it takes place in the same stage with the write to the HTRF caused by cell departures, which cannot be done any earlier. Performing the two write actions in corresponding pipeline stages, combined with the fact that the two operations are never initiated simultaneously, enables time sharing of a single memory port by both accesses.

Finally, it is not clear why we search the OutMask memory along with the VPout memory during credit arrival, in order to detect cells waiting for the credit. Reasoning this action is important, because it forces the OutMask memory to support content-addressable accesses. One may argue that VP/VCout matching is enough, but in this way, there may be a match between the credit's VP/VCout and a VPout entry containing "garbage" (the corresponding cell has departed and the slot is marked free). Still this is not the reason for OutMask searching, since this problem can be solved by concurrently searching the Free List block [4]. Not searching the OutMask memory may cause an error in the following scenario : suppose that a multicast cell of the flow-controlled group X is stored in the switch, it is at the head of its ready queue and needs to be transmitted to outlinks 3 and 4. The cell is transmitted to outlink 4 which is idle at the time, but not to outlink 3 because it is busy with higher priority traffic. If the credit for group X and outlink 4 returns to the switch before outlink 3 becomes available, and no OutMask search is performed, the credit will be given to the cell once more. Since the cell has already been transmitted to outlink 4, handing the credit to the cell is equal to losing it. It should be handed to the Credit Table where the next cell for that group can get it from. If the OutMask memory was searched as described above, there would be no matching, because the fourth bit in the cell's OutMask entry is cleared when it is transmitted to outlink 4. In addition, this also prevents matching with slots containing "garbage", since, when a cell is transmitted to its last destination and its slot is marked free, all bits in the corresponding OutMask entry have already been cleared. Hence, searching the OutMask memory is necessary for the correct operation of the credit-based flow control protocol.

[4] The Free-List block contains a small 256 x 1 memory, thus it does not cost much to turn it to a content-addressable memory (CAM)

4. Accesses Timing and Port Requirements

In the previous section, we described the accesses performed on each memory block, supposing that they have enough throughput to serve them all. Here, we investigate the number of ports per memory needed to achieve the required throughput for these accesses. In order to do this, we use the information on block operations and timing of events.

A cell-time is defined to be the minimum time period between two back-to-back cell arrivals or departures. A cell-time lasts 33 clock cycles, which is 660ns for the *ATLAS* I switch (20ns clock cycle). Within one cell-time, there can be up to one cell arrival, one cell departure and 2 credit arrivals per pair of links (input link X - output link X, X=0,1,...,15). Since, *ATLAS* is a 16 x 16 switch, the maximum number of events per cell-time is 16 cell arrivals, 16 cell departures and 32 credit arrivals [KSVMC96]. There is also the case of an additional event per cell-time, e.g. an extra cell arrival caused by a small difference in the clock periods of two neighboring switches, the injection/delivery of a cell through/to the Switch Control and Monitoring block or the execution of a management command. These events usually utilize the 33rd cycle or clock cycles during which the resources necessary are not occupied by normal events. Thus, serving such an additional event may be delayed for a few cell-times before they are properly served.

The *Queue Management* block (just as the whole switch) must have enough resources to be able to serve all normal events within one cell-time from their appearance. This is so in order to ensure that the correct data is written/read to/from the shared data buffer [KaVE95] and the size of FIFO memories in the credit/cell serialization blocks is kept small . Hence , each memory block must have enough throughput to accommodate for all the accesses that may be performed during the corresponding operations. Based on the frequency per cell-time of events and the operation of the block as described in the previous section, we can construct table 2, that analyzes the type [5], the frequency and timing of each access per event and per memory in the *QM* block.

4.1 Ports Calculation and Verification

Using table 2, we can calculate the number and type of ports per memory block. First of all, no port can accommodate for more than 33 accesses per cell-time (33 clock cycles). Thus, the minimum number of ports needed per memory can be calculated by dividing the number of accesses by 33 (or 32). Yet, we must also keep in mind the relative timing of the accesses, as defined by the pipeline of their operation and their timing relations to accesses performed to the same memory during different operations, in order to verify that this number of ports is sufficient.

Credit arrival operations may be initiated up to 32 times per cell-time, i.e. the credit arrival pipeline

[5] Type of accesses : rd read access; wr write access; sr search access; mod/rd modify/read access.

Event	Freq	Stage	VPout mem	OutMask mem	CreditMask mem	LinkList mem	Head-Tail reg. file
Cell Arrival	16	1					rd
		2	wr	wr	wr	wr	
		3					wr
Cell Departure	16	1					rd
		2	rd	mod/rd		rd	
		3					wr
Credit Arrival	32	1	sr	sr			
		2		rd	mod/rd		
		3					rd
		4				wr	wr
Total number of accesses			64	96	48	64	128

Table 2: Memory accesses per event in the *QM* block.

can be started on every 32 out of 33 clock cycles. Hence, each access performed by this pipeline needs a dedicated memory port, used on every 32 out of 33 cycles in worst case. On the other hand, cell arrival and departure operations may be initiated up to 16 times per cell-time each, they have the same number of pipeline stages and, therefore, can be both served by a single pipeline (as mentioned earlier). This imposes the restriction that only one operation, either for cell arrival or cell departure, may start on every clock cycle, but this complies with the operation of the shared data buffer [KaVE95] (either a read or a write operation may start on a single cycle). A closer look at table 2 also reveals that the two operations perform accesses to the same memory block in corresponding stages. Thus, time sharing memory ports between the two operations is possible and safe, since only one of the two operations may be in a certain pipeline stage on each clock cycle. Consequently, we actually need one memory port for every couple of accesses to the same memory performed by the pipeline, one for a cell arrival and one for a cell departure operation in corresponding stages. Taking into account the aforementioned observations and table 2, we calculate the actual number and type of ports per memory needed. This information, along with the explanation of the ports' use, is shown in table 3. One can notice, that the number of ports is the minimum. This is the result of carefully matching the actions and their timing for cell arrival and departure operations.

4.1 Ports Calculation and Verification

Memory block	Number of ports	type of each port	usage
VPout	2	read/write	read/write VP/VCout field on cell I/O
		search	search VP/VCout field on credit arrival
OutMask	3	read/write	write/modify-read outmask field on cell I/O
		search	search for enabled outlink on credit arrival
		read	read outmask field on credit arrival
CreditMask	2	write	write creditmask field on cell arrival
		read/write	modify-read creditmask field on credit arrival
LinkList	2	read/write	read/write next_cell_pointer on cell I/O
		write	write next_cell_pointer on credit arrival
Head/Tail	4	read (1)	read head/tail pointers on cell I/O
		read (2)	read head/tail pointers on credit arrival
		write (1)	write head/tail pointers on cell I/O
		write (2)	write head/tail pointers on credit arrival

Table 3: Ports required and their purpose per memory block.

4.2 HTRF alternative organizations

The Head-Tail register file seems to be the most demanding, in terms of throughput, memory, since it needs to be four-ported. A few other alternative organizations for the information provided by this block were examined, in order to reduce the number of ports and avoid designing a four-ported SRAM. The first alternative was to place the head and tail pointers in separate memories. Although it seemed, at first sight, that both memories would be three-ported, operations on empty queues, where enqueueing demands writing both head and tail pointers, raised the number of ports for the tail register file to four. Since we would still have to design a four-ported SRAM and, in addition, a three-ported one, this alternative was abandoned. The second alternative examined, was to keep head and tail pointers separated and duplicate the tail register file as well. While write accesses to tail pointers would be performed by both register files, a read access could be served by any single one of them. In this way, all three register files would be three-ported. Yet, the chip area wasted and the additional complexity of control and bypass logic, made this solution unattractive too. After all, Head-Tail register file is a small memory (54 x 17) and must operate at a rather low speed for its size (20ns cycle time), hence we expect that it will not be that difficult to design a four-ported SRAM cell for it.

5. Queue Management Control

Control units in switch blocks are usually built as finite state machines (FSM). FSMs provide a simple way to encode the state of the block and produce control signals. In addition, their gate-level circuit can be automatically derived from behavioral descriptions. Yet, the use of a single FSM for the control unit of the *Queue Management* block is practically impossible, because of its pipelined and superscalar operation. Since, the credits pipeline has four stages and the cells pipeline has 3, each one operating on either a cell arrival or departure, a single FSM would have 432 (!) different states.

The method employed for the *QM* control unit is the one used for pipelined and superscalar CPUs [PaHe93]. Once a pipeline is initiated, all the necessary control signals are calculated on the first stage. They are transferred to the following stages through pipeline registers, until they are "consumed" by the proper stage. Naturally, on every stage, the information available at the time is used in order to verify the correctness of the control signals and to selectively alter or cancel some of them, if necessary. Two such control units exist within the *QM* block: one for the credits pipeline and one for the cells pipeline. In the following paragraphs, details about the two control units and their operation are presented. For better comprehension of their operation, one must have in mind the operation of the whole block on the various events, as described in section 3.

5.1 Credits Pipeline Control

The control unit of the credits pipeline is responsible for the control signals necessary during credit arrival operations. The original generation of these signals is triggered by the Credit Serialization block. Figure 3 presents the flow-control diagram of the unit. The control signals presented in figures 1 and 2, that are asserted in each stage, are shown inside the stage symbol. The arcs between the stages define the next stage, and are labeled with conditions that select a specific next stage, when multiple next stages are possible. As mentioned earlier, the pipeline may be initiated every 32 out of 33 clock cycles in a cell-time, so multiple stages may be active simultaneously.

In stage 1 (search stage), the search enable signals for the VPout and OutMask memories are activated. At the end of the stage, the result of the search action (signal credit_match) is used to decide whether control will flow to the next stages, or the credit will be handed to the Credit Table block. Stage 2 (read & compare stage) sets the CRmod/re signal; this forces a modify access to one bit and a read access to other 15 bits in the CreditMask entry of the cell receiving the credit, through the corresponding port. Asserting the OMre signal causes a read access to the corresponding OutMask entry. The results of these two accesses are compare to detect if the cell is ready. In stage 3 (read tail stage), asserting HTVre2 triggers an access to the HTRF from the second read port, for the tail pointer and valid bit to be read. Finally, in stage 4 (enqueue stage), control depends on whether the queue is empty at the time or

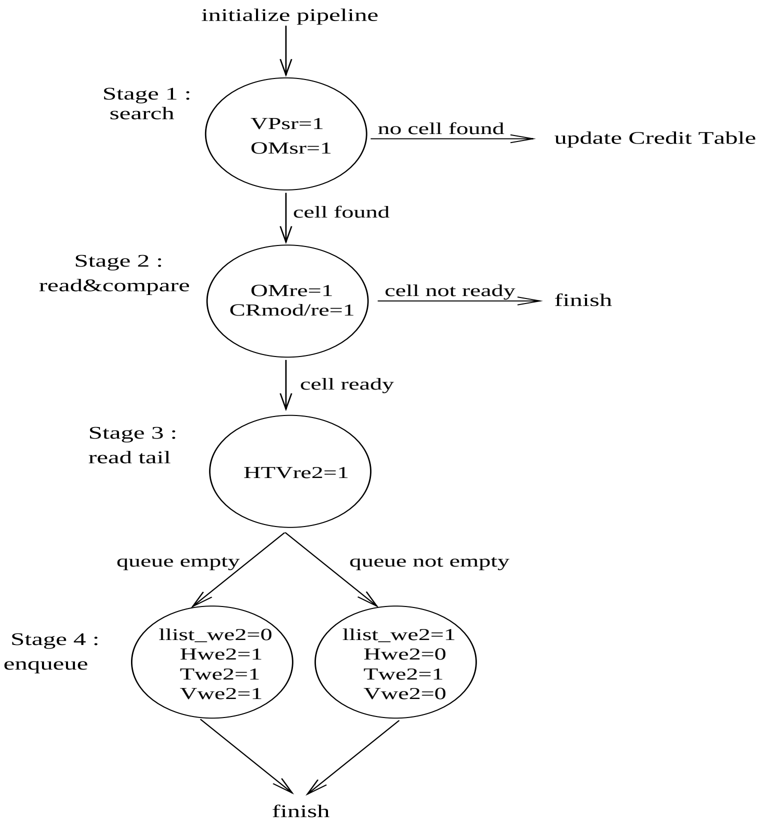


Figure 3: Flow-Control diagram for the credits pipeline

not. In the second case, llist_we2 is set to cause a write access to the next_cell pointer of the current tail in the LinkList memory, while asserting Twe2 causes the tail pointer to be updated through the second write port of the HTRF. In the first case, llist_we2 is reset; Hwe2 and Vwe2 are set to force changing the head pointer and the valid bit of the queue through the same HTRF port, as well.

5.2 Cells Pipeline Control

The control unit of the cells pipeline serves a dual function; it generates control signals for both cell arrival and departure operations. Its flow-control diagram follows the two control streams in figure 4. During a clock cycle, only one of two corresponding stages from the two streams may be active, since either a cell arrival or departure operation may be initiated on a certain cycle by the Scheduler block. Yet, more than one, not corresponding, stages can be active simultaneously from either streams.

The most important signal of the unit is the one that distinguishes cell arrival from cell departure operations. It is actually generated by the Scheduler block and is propagated through the pipeline stages, designating whether the stage should execute the cell arrival or departure actions. This signal (out/in_),

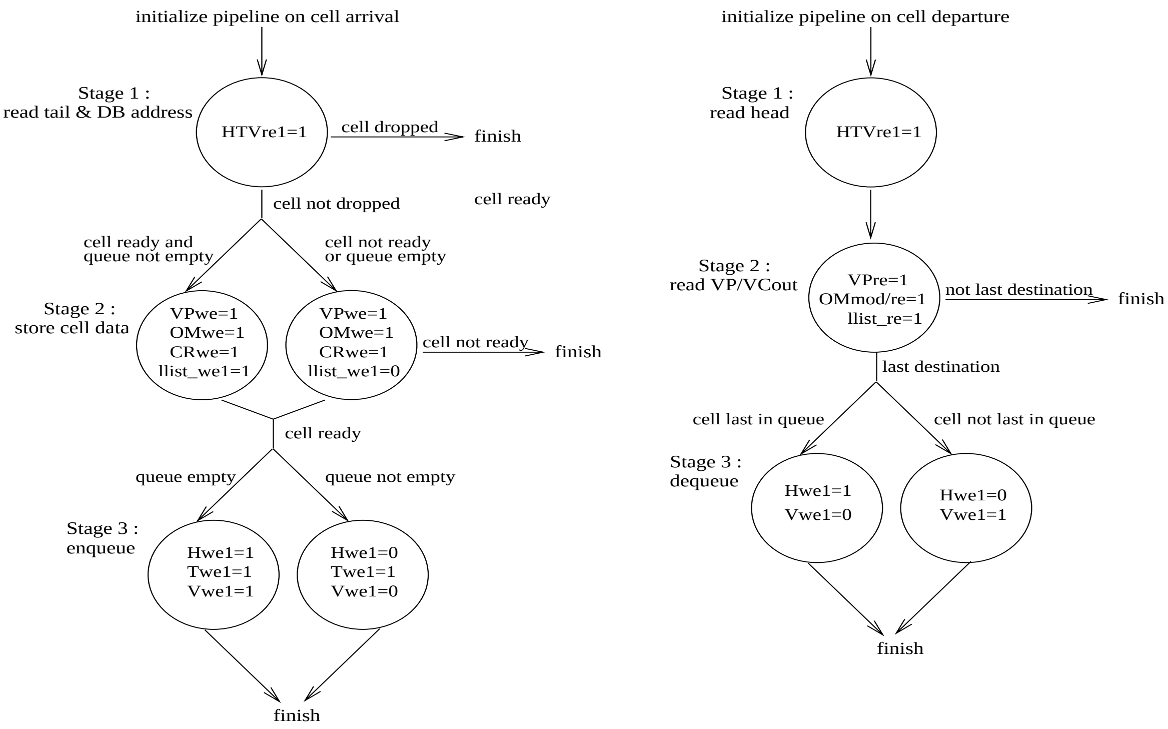


Figure 4: Flow-Control diagram for the cells pipeline

along with its delayed by one clock cycle version (out/in_2), controls a number of multiplexors, that select and prepare the correct input data and addresses for the accesses of the two operations. Thus, out/in_ coordinates the time sharing of the pipeline resources by the two operations.

For cell arrival operations, control flows according to the left stream in figure 4. In the first stage (read tail & DB address stage), assertion of HTVre1 triggers an access through the first read port of the HTRF, in order to read the tail pointer and valid bit. The data received from the Free List and Cell Counters block are used to decide if the cell is to be dropped or not. Stage 2 (store cell data stage) sets the write enable signals for the VPout, OutMask and CreditMask memories (VPwe, OMwe and CRwe respectively), so that cell information is properly stored. If the cell is ready and the queue is not empty, llist_we1 is set as well, so that the next_cell pointer of the current tail is written through the read-write port of the LinkList memory. The appropriate address for this access, which is the previously read tail pointer, is selected by the out/in_2 signal. Note that this is not the same with the address for the accesses to the VPout, OutMask and CreditMask memories (celio_adr), which identifies the slot to which the incoming cell is stored. If the cell is not ready, the control flow terminates. Asserting Twe1 during stage 3 (enqueue stage), forces the DB address of the cell (freeslot_enc) to be written in the tail pointer entry through the first write port of the HTRF. If the queue is empty, Hwe1 and Vwe1 are also set for the head pointer and valid bit to be written.

5.2 Cells Pipeline Control

Control for cell departure operations follows the right stream in figure 4. At stage 1 (read head stage), setting HTVre1 forces the head and tail pointers to be read through the first read port of the HTRF. Reading the VP/VCout field of the cell is succeeded by asserting the VPre signal in stage 2. At the same time, the read/write port of the LinkList memory is used to read the pointer to the next cell in the queue (llist_re set). Finally, a modify-read access is triggered in the the OutMask memory, by setting OMmod/re and selecting the correct input data with the out/in_ signal. If the 15 bits read are all zero, the cell is currently transmitted to each last destination and, therefore, dequeue stage (stage 3) must follow. In the case that the cell departing is not the last one in the queue, asserting Hwe1 forces the next cell pointer just read to be selected by the out/in_2 signal and written in the HTRF as the new head pointer (using the second write port). Otherwise, the Vwe1 signal is set, so that the valid bit of the queue is reset.

6. Bypass Control and Datapaths

Superscalar and pipelined processing naturally involves data hazards [PaHe93][PaHe95]. The data used for calculations by the first pipeline may be simultaneously altered by the second one. The same situation can also come up between different stages of the same pipeline. In order to confront with this problem, one must first locate the sources of data hazards and the conditions under which they arise, and then provide for a stable solution. Two methods for solving this problem are usually employed [PaHe93]. The first one, stalling a pipeline whenever a data hazard occurs, is unacceptable for the *ATLAS* I switch, since stalling would result to either dropping incoming cells and credits, or underutilizing the output throughput. With the second solution, called bypassing (or forwarding) results, one provides additional propagation paths for the data in order to make them available to the rest of the block as soon as they are calculated, without waiting for them to be transferred to their regular position.

There are two sources of hazards in the *Queue Management* block. The first one, and the easiest to deal with, is caused by concurrent arrivals of cells and their corresponding credits. The second source is concurrent or back-to-back operations to the same ready queue, which may lead to use of mistaken head and tail pointers. In the rest of this section, we examine the hazard conditions for both cases and present the bypass datapaths and control necessary for correct operation.

6.1 Cell/Credit Arrival Bypass

In the first stage of cell arrival operations, the Credit Table block is accessed for available credits for the incoming cell. Similarly, in the first stage of credit arrival operations, the creditless cell list is searched for cells waiting for the incoming credit. Yet, if the two pipelined operations for the arrivals of a cell and its corresponding credit overlap, it is possible that they will "miss" each other. In other words, the cell will not receive the credit from the Credit Table, while the search action during the credit arrival will fail. As a consequence, the cell will be blocked, along with all the following cells of its flow-group.

Figure 5 presents the possible overlapping between the pipelines serving the arrivals of a credit and its corresponding cell, along with the situations when hazards occur. Boxes of the cell arrival pipeline describe the actions for a not ready cell, while the credit arrival pipeline depicts the actions for a credit expected by no cell in the CLL.

In situation 1, the credits pipeline searches for cells waiting for the credit at the same time the cells pipeline stores the cell information in the block memories. Since one cannot guarantee which value of the memory entries written, the old or the new one, will actually be compared to the search pattern, it is safer to consider that this entries will not match [6]. Thus, situation 1 conceals a hazard case. The same holds for situation 2, where the search action takes place one clock cycle before the cell information is

[6] Match canceling is accomplished by pulling down the match lines of memory words written during the same clock cycle.

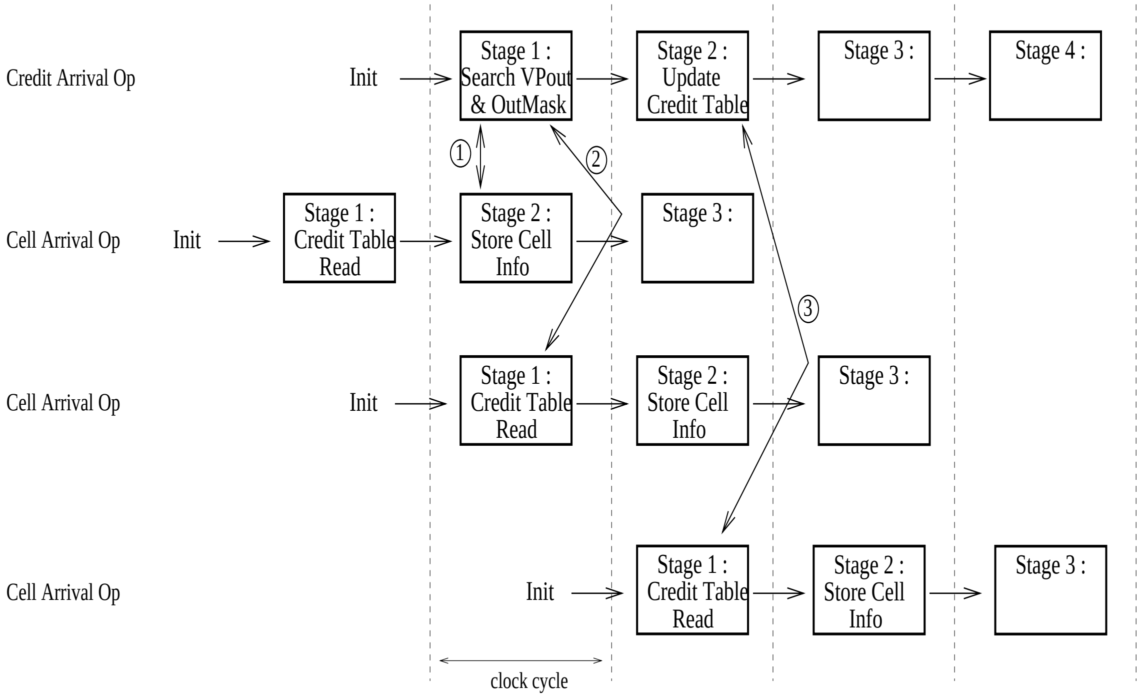

Credit Arrival Op
Init
Stage 1 : Search VPout & OutMask
Stage 2 : Update Credit Table
Stage 3 :
Stage 4 :
1
2
Cell Arrival Op
Init
Stage 1 : Credit Table Read
Stage 2 : Store Cell Info
Stage 3 :
3
Cell Arrival Op
Init
Stage 1 : Credit Table Read
Stage 2 : Store Cell Info
Stage 3 :
Cell Arrival Op
Init
Stage 1 : Credit Table Read
Stage 2 : Store Cell Info
Stage 3 :
clock cycle

Figure 5: Data hazards between credit and cell arrival operations.

stored, while the read access to the Credit Table by the cell arrival pipeline is performed even before the credit starts being processed by the *QM* block. On the other hand, no hazard exists in situation 3, if writing and reading the same entry of the Credit Table presents the new data to the memory output [7].

Bypass datapaths for situations 1 and 2 are provided in the diagram of the *QM* block (figure 1). For situation 1, the hazard is avoided by keeping the decoded DB address of the incoming cell instead of the mask indicating the search result. In this way, the search access successfully detects the incoming cell. The necessary multiplexor is being controlled by the crd_bypass_1 signal, asserted when the incoming cell served by the second stage of the cells pipeline belongs to the same flow-group with the credit currently at the first stage of its pipeline. Situation 2 is handled by setting the bit corresponding to the arrival link of the credit (indicated by crin_link_2 vector) in the mask of available credits for the incoming cell (celiCRmask). The rest actions of the credits pipeline are canceled in order to avoid transferring the credit to the Credit Table as well. This bypass action is controlled by the crd_bypass_2 signal, which is set whenever the incoming cell and credit, being served by the two pipelines in their first stages, belong to the same flow-group.

[7] This can be handled by the peripheral circuits of the Credit Table memory without any modifications to its basic memory cell.

6.2 Head-Tail Pointers Bypass

Cell and credit events change the status of the ready queues and the creditless list. While the latter is kept as a pool of cells, thus no special structure or connectivity is maintained, the ready queues are kept as linked lists. The status of these lists is kept in the form of the head-tail pointers and the valid bits, all stored in the HTRF memory, and is updated every time a cell is added in the list (on a credit or cell arrival), or a cell is removed from it (on a cell departure). References and updates to this information must always be executed in the "correct" order in order to preserve the connectivity of the list. Failure to maintain this order may lead to reading or writing wrong pointers, and consequently, either losing some cells and blocking their corresponding flow-groups as well (if they are flow-controlled), or creating non existing cells.

Changes in the queue status are performed in two stages on all events. In an initial stage, the current queue status is read from the HTRF. This is later used, along with the data about the credit or cell served, to calculate the new queue status, which is written back to the HTRF in a following cycle. This time difference between reading the current status and writing the new one, may be one or two clock cycles, hence it is possible for another action to also modify or read the queue status in the meanwhile. One must make sure that each of the two or more actions operating on the same queue concurrently will read and write the correct data in a well defined order, so that the queue structure remains consistent.

The normal way of working through this problem would be to try to spot all the possible cases of hazard caused by concurrent operations on the same queue. Yet, the possible valid timing combinations of the three events and different queue states (empty, one cell in, more than one cells in), measure up to 482(!), as proven by simulations conducted, thus examining them one by one is practically impossible. For this reason, an alternative strategy is used. First of all, case analysis is performed for every one of the six possible combinations of two concurrent events, in order to spot simple hazard conditions and the bypass rules necessary to handle them. For each operation, analysis is confined only to stages between the one where the queue status is read and the one where the new status is written. This reduces credit arrival operations to only two stages, while cell arrival and departure operations remain three stages long. The analysis for the case of possible concurrent executions of one cell arrival and one credit arrival operation, both working on the same ready queue, is shown in figure 6. Assuming that the same data can be written and read in the same clock cycle in the HTRF, only two situations of hazard exist. In situation 1, the cell receiving the credit will actually be enqueued before the incoming cell, thus the tail pointer and valid bit read by the incoming cell in stage 1, must be bypassed values from the credits pipeline. On the other hand, in situation 2, both cells are enqueued in the same clock cycle. We choose to have the cell receiving the credit last, since bypassing the tail pointer and valid bit from the cells pipeline to the credits pipeline is feasible [8]. Apart from the bypass, the write access to the HTRF performed by the cell

[8] The cell arrival pipeline is already using the tail pointer and valid bit, at the time the situation is detected.

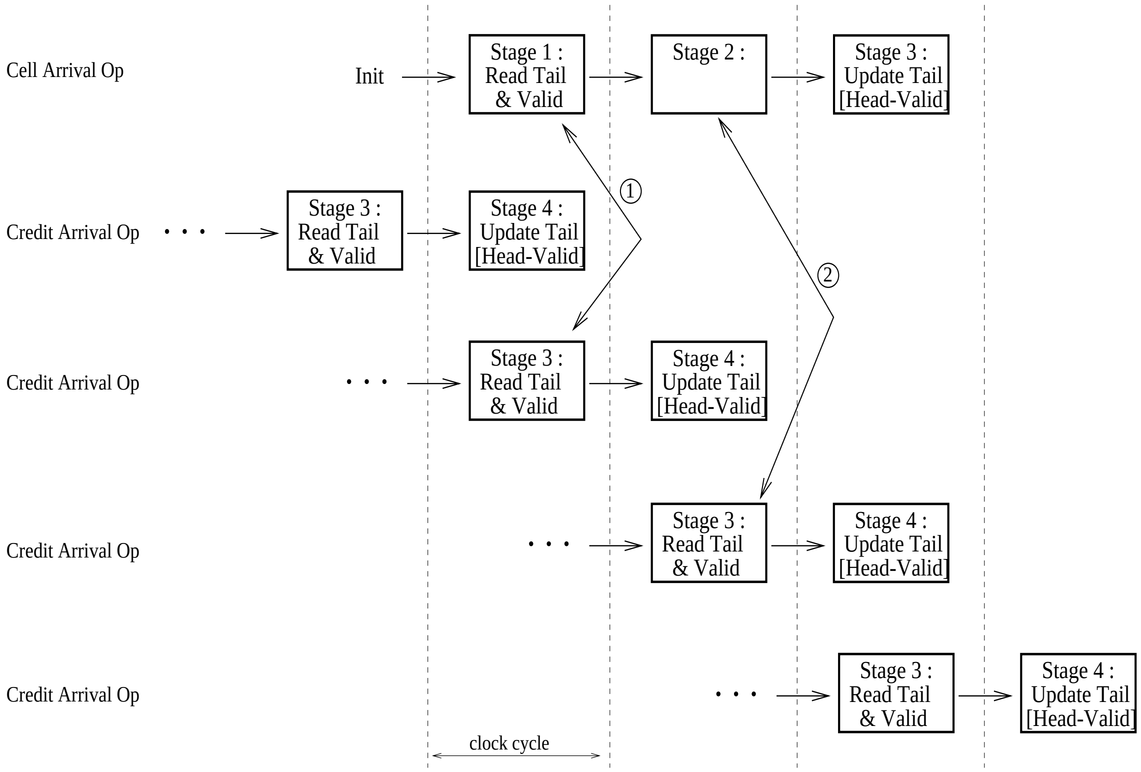

Cell Arrival Op
Init
Stage 1 : Read Tail & Valid
Stage 2 :
Stage 3 : Update Tail [Head-Valid]
Credit Arrival Op
Stage 3 : Read Tail & Valid
Stage 4 : Update Tail [Head-Valid]
1
2
Credit Arrival Op
Stage 3 : Read Tail & Valid
Stage 4 : Update Tail [Head-Valid]
Credit Arrival Op
Stage 3 : Read Tail & Valid
Stage 4 : Update Tail [Head-Valid]
Credit Arrival Op
Stage 3 : Read Tail & Valid
Stage 4 : Update Tail [Head-Valid]
clock cycle

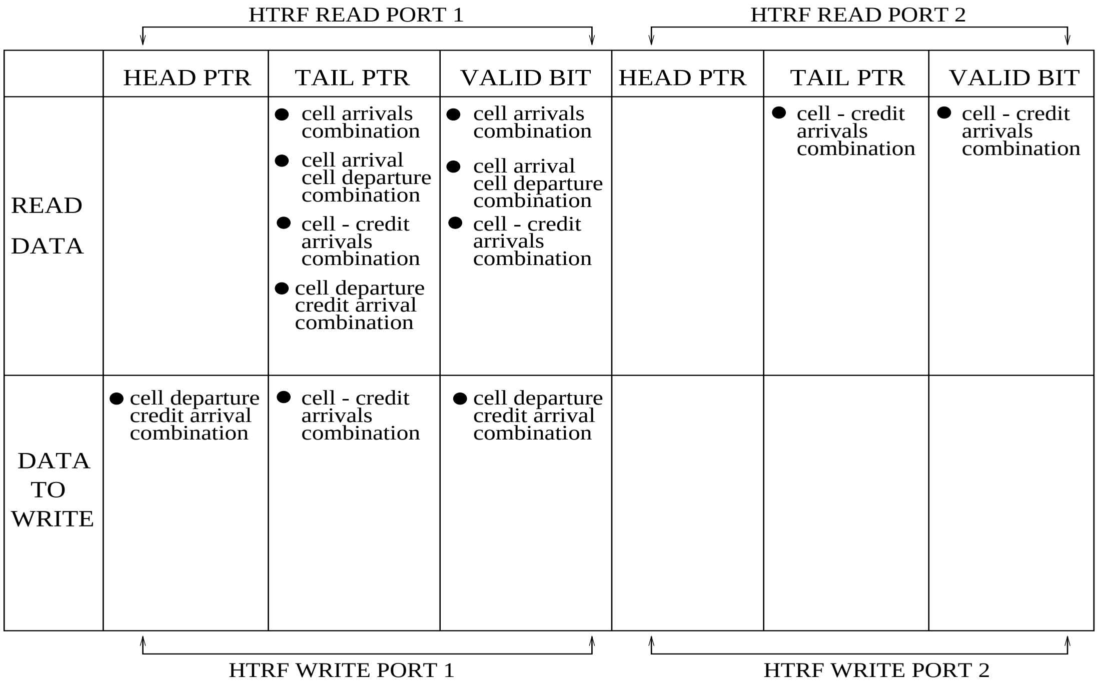
Figure 6: Data hazards from concurrent cell and credit arrival operations on the same ready queue.

pipeline is canceled.

By performing such an analysis for each pair of concurrent events, table 7, which summarizes the hazard cases for the data read and written through the four ports of the HTRF and the combination of events that causes them, can be constructed. For values read or written, that only one hazard case exists, the bypass condition and rule is the one indicated by the simple case analysis. On the other head, for the tail pointer and valid bit read through read port 1, for which multiple hazard cases exist, all possible combinations of cases must be checked as well. Yet, some combinations are not valid, since they assume concurrent initiation of a cell arrival and a cell departure operation. Hence, only three combinational cases exist for each value, and it is easy to define bypass condition and rules for them too (by conducting detailed case analysis).

The above described procedure results to the bypass conditions and sources summarized in table 4. The last column states the bypass condition in terms of the stages in the two pipelines that have to be active and operating on the same queue. For the cells pipeline, the kind of operation served by the stage is stated in parenthesis. The value forwarded is given in a coded form [9] in the third column. In addition

[9] CrC : DB address of the cell receiving the incoming credit; InC : DB address of the incoming cell; Cancel : write access



HTRF READ PORT 1

HTRF READ PORT 2

	HEAD PTR	TAIL PTR	VALID BIT	HEAD PTR	TAIL PTR	VALID BIT
READ DATA		● cell arrivals combination ● cell arrival cell departure combination ● cell - credit arrivals combination ● cell departure credit arrival combination	● cell arrivals combination ● cell arrival cell departure combination ● cell - credit arrivals combination		● cell - credit arrivals combination	● cell - credit arrivals combination
DATA TO WRITE	● cell departure credit arrival combination	● cell - credit arrivals combination	● cell departure credit arrival combination			

HTRF WRITE PORT 1

HTRF WRITE PORT 2

Figure 7: Summary of the hazard situations for the data read and written to the HTRF.

to these rules, we assume that reading and writing a HTRF entry in the same clock cycle, produces the new value as read result. These rules are implemented through the multiplexors placed at the inputs and outputs of the HTRF (shown in figure 2). Their control signals are produced by combinatorial logic, a few gates and equality comparators, that detects the bypass conditions and activates the proper bypass path each time.

6.3 Bypass rules and datapath verification

As mentioned earlier, maintaining the correct connectivity for the ready queues is crucial for the operation of the switch. In addition to that, we expect that concurrent events on the same queue will be an often case when switch the operates under heavy load, bundled links are used or most of the traffic is destined to a limited number of destinations in the network. In other words, we expect that bypass conditions will appear frequently. Since bypass rules properly maintain the queues' connectivity on concurrent events, their correctness needs to be verified through extensive simulation, before hardware implementation.

Two functional models were written for simulation use. They both receive cell and credit events as inputs and maintain the status of a single ready queue. This includes head and tail pointers, the valid bit and one pointer per cell in the queue, in order to build the linked list structure. The first model, called the

is being canceled.

HTRF Data	Prio	Bypass Value	Bypass Condition
Tail Ptr 1 (Read)	1	CrC	Cells Stage 1 (any); Credits Stage 3
	2	InC	Cells Stage 1 (any) and 2 (arrival)
Valid Bit 1 (Read)	1	1	Cells Stage 1 (arrival); Credits Stage 3
	2	1	Cells Stages 1 (arrival) and 2 (arrival)
	3	0	Cells Stage 1 (arrival) and 2 (departure); departing cell last
Tail Ptr 2 (Read)		CrC	Cells Stage 2 (arrival); Credits Stage 3
Valid Bit 2 (Read)		1	Cells Stage 2 (arrival); Credits Stage 3
Head Ptr 1 (Write)		CrC	Cells Stage 2 (departure); Credits Stage 3; departing cell last
Tail Ptr 1 (Write)		Cancel	Cells Stages 2 (arrival); Credits Stage 3
Valid Bit 1 (Write)		1	Cells Stages 2 (departure); Credits Stage 3

Table 4: Bypass conditions and sources, according to priority (when multiple sources exist), for data read and written in the HTRF.

real-model, updates this information on each enqueue or dequeue operation caused by an event, in the same way this is performed by the *Queue Management* block. This means that the model follows both the pipelined operation and the bypass rules of the block. On the other hand, the second model, called sim-model, maintains the status information by serving all events on a single clock cycle. Events starting concurrently are served on a fixed order, credit events first and cell events second. There are no hazard cases and no need for bypassing in the sim-model. Verification of the bypass rules is accomplished by feeding the same events to both models and comparing the two queues at regular intervals. Mismatches between the status or the connectivity of the queues, indicates that certain bypass rule is either incorrect or missing.

The models were used with two simulation methods. The first one performs random tests. Random events are generated and fed to the models. The appearance probability of each event is controlled by uniform random variables within segments of parameterized length. On regular intervals, the generation of events is interrupted and the queues maintained by the two models are compared. Millions of simulated clock cycles for the two models were executed with this method.

Although extensive testing with random patterns should probably reveal most errors, still one can not be sure that all possible error situations are examined. In order to make sure that no combination of events was left without being tested, we developed the second simulation method. In this method, we create and test automatically all valid combinations of active pipeline stages and queue states (empty, one cell in the queue, many cells in the queue) on a certain clock cycle. Since the credits pipeline has four stages and the cells pipeline has 3 stages with dual role, there are $2^4 * 3^3 = 432$ possible combinations of active stages on a clock cycle. Each one of them, if it is a valid one, is created for the two models for each of the three queue status. The combinations of stages are created by generating the proper events within a few cycles, while the queues are initiated to contain the desired number of cells each time. After a few clock cycles, used by the real-model to serve the events, the two queues are compared for mismatches. In this way, not only we can detect errors in the bypass rules but specifically spot the cases incorrectly handled as well. Once the simulation of the two models with this method was successfully performed, the correctness of bypass rules was assured.

7. Management Commands Support

Apart from the normal operations, the *Queue Management* block must also support management commands. By using these commands, we must be able to read and write every memory entry in the block. The purpose of their existence is to enable testing of the functionality of the block, proper initialization of the memories and the execution of the algorithm for lost cells/credits detection [KSVMC96].

All blocks exchange data and addresses for management commands through two busses that run across the whole switch, and connect the various blocks with the Switch Control and Monitoring block. The functionality of this block is to receive management commands from incoming cells or through the Test/Configuration port, forward them to the appropriate block within the switch and return their results to the proper destination. The first bus, C_bus, is 9 bits long and identifies the block and the register within it, from/to which data are read/written through the other bus. The second one, I_bus, is 16 bits long and transfers management data and commands to and from the various blocks.

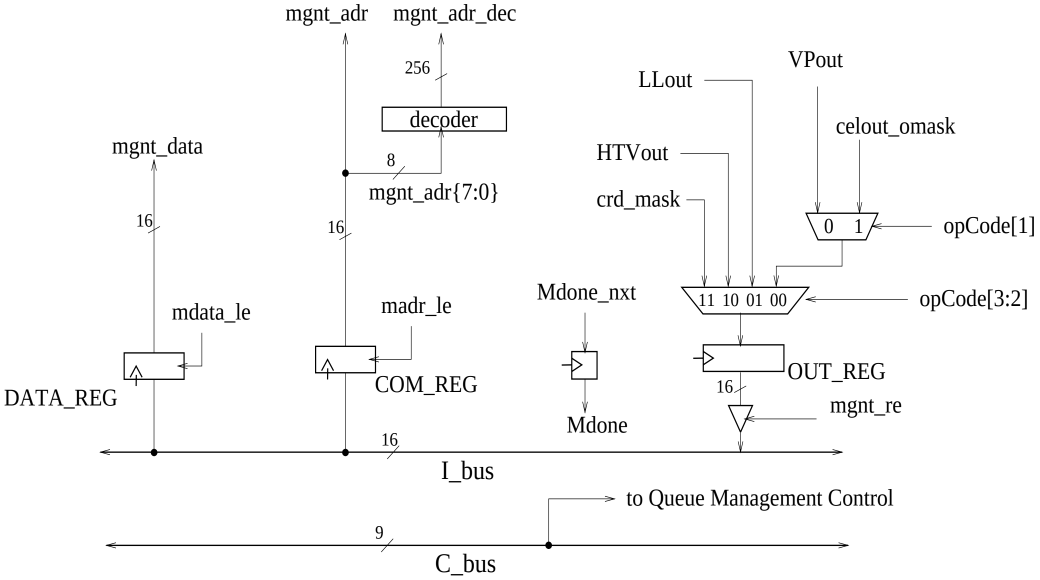


Figure 8: The *Queue Management* interface for management commands.

The *QM* block interface for management commands (figure 8) mainly consists of three registers. The DATA_REG register stores data for write accesses through management commands, while the COM_REG register keeps the address of the accesses and their opcode. OUT_REG register stores the data read through these commands. Each one of the three registers has a unique 9bit address within the switch. When one of these addresses appears on the C_bus, either data on the I_bus are stored in the DATA_REG or COM_REG register, or the contents of OUT_REG are driven on the I_bus. In the rest of this section, we explain the commands supported, their format and how they are served by the *QM* block.

7.1 Management Commands and their Format

Management commands are sent to the *QM* block by writing to the COM_REG register. Its contents are divided into five fields, presented in figure 9. Bits 7 to 0 contain the address for the memory access performed by command. For accesses to the HTRF, only bits 5 to 0 are actually used. Bits 11 to 8 serve as the opcode of the command. Bit number 12 is the trigger bit, which identifies whether the register contains an unserved and valid management command. This bit is read by the control circuits in order to schedule and serve the command. As soon as it is executed, this bit is reset. The extra bit (bit number 13) is used as the 17th data bit in write accesses to the HTRF and the CreditMask memory, because their word is one bit longer than the DATA_REG register. The last two (most-significant) bits are not used.

15 14	13	12	11 8	7 0
Unused	Ex	Tr	opCode	Address

Tr = Trigger Bit
Ex = Extra Bit

Figure 9: Command Register (COM_REG) fields.

The *Queue Management* block recognizes the following commands, which are also summarized for convenience in table 5, along with their opcode and required data:

VPout Read This command reads one of the 256 entries in the VPout memory. The access is performed through its read/write port, normally used by the cells pipeline.

VPout Write Writes an entry in the VPout memory. The data written are the 12 least-significant bits (11-0) in the DATA_REG register. It uses the same port with the corresponding read command.

OutMask Read Reads an entry in the OutMask memory. It is performed through the read/write port, used for cell arrival and departure operations. All bits in the DATA_REG register must be reset before the access is performed.

OutMask Write This command writes an entry in the OutMask memory. The contents of the DATA_REG register are used as write data. It is served from the same memory port with the OutMask Read command.

CreditMask Read/Modify Using this command one can set some bits while reading the rest in a CreditMask memory entry. Bits to be set are indicated by ones in the corresponding bits of the

DATA_REG register and the extra bit. The new values of the bits set are also presented with the read data. The access is served through read/write port of the CreditMask memory, used by the credits pipeline during usual operation. In order to perform a plain read access, all bits in the DATA_REG register, as well as the extra bit in the COM_REG register, must be reset.

LinkList Read Reads an entry from the LinkList memory. The access uses its read/write port, normally used by the cells pipeline.

LinkList Write Writes an entry in the LinkList memory. The 12 least-significant bits in the DATA_REG register are used as input data. It uses the same port with the corresponding read command.

HTRF Read Reads a word from the Head-Tail Register File (HTRF). It is performed through its second read port, normally used for credit arrival operations.

HTRF Write This command writes a word in the HTRF. The contents of the DATA_REG register as well as the extra bit, are used as write data. The access uses the second write port of the register file, normally used by the credits pipeline.

	Memory	Access	opCode	Data required
1	VPout	Read	0000	no data necessary
2	VPout	Write	0001	DATA_REG (12 LS bits)
3	OutMask	Read	0010	DATA_REG=0
4	OutMask	Write	0011	DATA_REG
5	CreditMask	Read/Modify	1101	DATA_REG plus Extra bit=modify_mask
6	LinkList	Read	0100	no data necessary
7	LinkList	Write	0101	DATA_REG (12 LS bits)
8	HTRF	Read	1010	no data necessary
9	HTRF	Write	1011	DATA_REG plus Extra bit

Table 5: *QM* management commands (opCode and required data).

7.2 Implementation of Management Commands

Management Commands are not served by the *QM* block as soon as they are issued. This would not be possible to achieve without either stalling the normal block operation or providing additional memory ports for their accesses. Neither effect is desirable. Memory blocks are already multiported and complex. In addition, the algorithm for lost credits/cells detection, that uses management commands, may be executed every few seconds, thus stalling the switch for its execution would be unacceptable. The commands are served as soon as the memory port they need is not used by a cell or credit operation. Hence, in the worst case, a management command will be served 32 cycles after it reaches the *QM* block, since at least one of the 33rd clock cycles within a few cell-times is not necessary for any normal operation. The circuit serving management commands in the *QM* block works as described in the following paragraphs.

Whenever the trigger bit in the COM_REG is set, the opCode field, along with the memory ports status, are used to detect when this command can be served. The necessary port is available if the pipeline stage, within which it is used, is inactive. As soon as it is detected that the proper memory port will be unoccupied in the next clock cycle, the management data and address, either in a decoded or encoded form, are fed to the corresponding memory. This is done with one of the management access signals (VP/OMmac, CRmac, LLmac and HTVmac), shown in figures 1 and 2. In the following cycle, the memory control signal corresponding to the access is asserted and the trigger bit is reset.

Data produced during read commands are selectively transferred to the OUT_REG register, through two multiplexors, controlled by the command's opCode (shown in figure 8). If the data read are less than 16 bits, they occupy the least significant bits of the register. On the other hand, if the data read are 17 bits long (HTRF and CreditMask memory), bit number 0 (least significant) is discarded.

When the access and the transfer of the results to the OUT_REG register are complete, the Mdone signal is pulled down for one clock cycle. This notifies the Switch Control and Monitoring block that the command has been successfully executed, and that the result data (if any) are available in the OUT_REG register. It is up to this block to read them before they are overwritten by the results of a second command. As soos as Mdone is pulled up again, the *QM* block can accept a new management command.

8. Block Functional Simulation and Testing

Functional simulation and testing of the *Queue Management* block has been performed, in order to ensure its correct operation, prior to its VLSI implementation. In the previous sections, we presented a number of conclusions, operations and methods, whose validity is easier tested through functional simulation than layout or gate level design checks. Extensive testing can reveal a wide range of errors, from architectural errors to incorrect synchronization or timing between different actions of the same operation, as well as certain aspects of the block architecture and operation that are so far neglected.

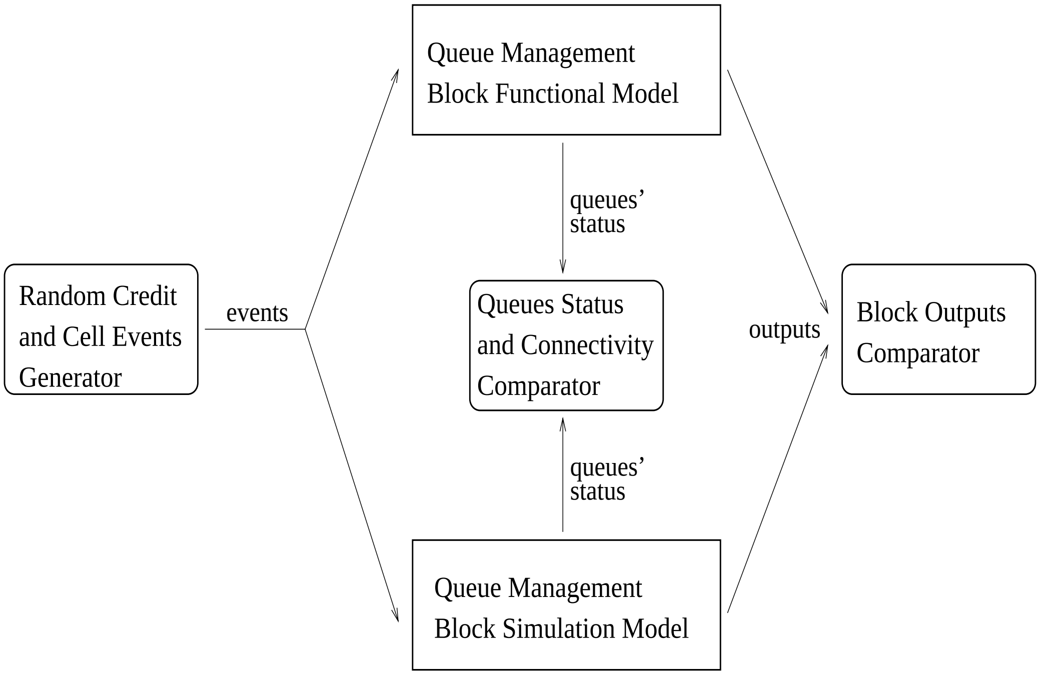

Queue Management Block Functional Model
queues' status
Random Credit and Cell Events Generator
events
Queues Status and Connectivity Comparator
outputs
Block Outputs Comparator
queues' status
Queue Management Block Simulation Model

Figure 10: The organization used for functional simulation and testing of the *QM* block.

The organization used for the simulation is presented in figure 10. The *QM* block functional model, written in Verilog-XL [Veri94] as all the rest of the code for this simulation, implements all the operations and functions described in the previous sections, with a clock cycle precision. This means that sub-blocks described in it, such as memories, decoders or control units, are these that will actually be designed, and were written in a behavioral manner so that they execute on each clock cycle the same operation with the real hardware. Control logic is described in a strictly behavioral manner, since this is easier to do when changes are frequent, and one can get excellent gate-level netlist from this description by using sophisticated synthesis tools. On the other hand, the *QM* block simulation model is a behavioral high-level description of the block, that executes all operations within a single clock cycle. Consequently, sub-blocks in this model do not match the real hardware, but all the necessary information in order to perform the same operations is maintained. Testing the *QM* block is achieved by regularly comparing the operations and status of the two models.

The random credit and cell events generator model simulates the rest of the switch by feeding events and inputs to the two models with the appropriate timing. The operation of all the switch blocks interacting with the *QM* block is taken into account in this model, so that the type and the timing of input signals is correct. Events generation is random; yet, the appearance probability of each event and its characteristics are parameterized. In this way, we can control the number cells entering or leaving the block, their flow-group, whether they have all credits available on their arrival, etc. This makes it possible to run a number of different simulations and test the block under various conditions, loads and types of traffic. There are also two comparator models. The outputs comparator continuously watches all outputs of the two models, such as the not_empty and reserve masks, and interrupts the simulation when they differ. The status and connectivity comparator is used in regular intervals in order to compare the memory contents of the *QM* models, and make sure that the queues maintained are the same and do follow basic rules (such as no list becomes a cyclic one). Whenever this comparator is used, events generation is temporally stalled. With the aforementioned simulation models, extensive testing to the *QM* block has been performed (millions of simulated clock cycles of operation).

Management commands where tested by extending the events generator to produce such commands as well. The outputs comparator was also extended to be able to detect correct completion of these commands. Yet, management commands changing the queues' status, i.e. write or modify commands, were also tested by injecting a few predetermined commands by hand, since random generation of such commands could destroy the correct connectivity of the ready queues.

Apart from the tests conducted with the above organization, the *Queue Management* block functional model will also be tested in the functional simulation of the whole *ATLAS* I switch. In addition to further undetected errors, these simulations will also check the interoperability and synchronization of the block with the rest of the switch.

9. Block Hardware (VLSI) Implementation

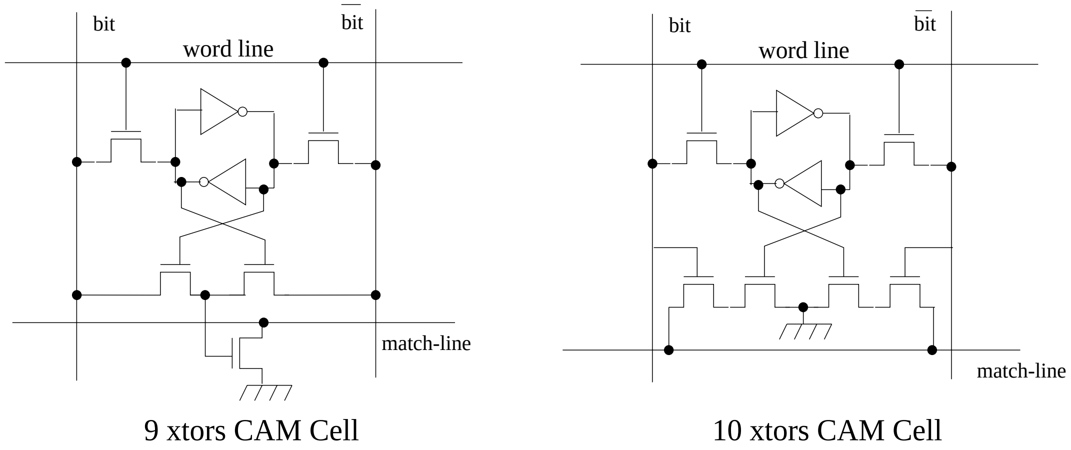
The *Queue Management* block is currently designed using both full-custom and semi-custom VLSI techniques. The target technology is the SGS-Thomsom Microelectronics 0.5μm CMOS technology with three metal layers and one polysilicon layer, operating at a supply voltage of 3.3 volts.

The five memories included in the block are multi-ported and have critical timing requirements and, therefore, need to be designed with full-custom mask-level layout techniques [WeEs93]. They all are static memories. On the other hand, control logic and the rest of the block logic will be designed with semi-custom layout techniques. Semi-custom gate-level layout can be produced automatically from functional or behavioral descriptions by using synthesis tools, such as Synopsys [Syn94].

In this section, we present the two-ported memories and their peripheral circuits already laid out in full-custom CMOS, and also describe the VLSI implementation of the remaining static memory blocks.

9.1 Content-Addressable Memory Cells

The VPout and OutMask memory blocks have to be content-addressable (CAM) [Gros92] in order to accommodate for the search action in the first stage of credit arrival operations. There are two basic alternatives in the layout of static CAM cells [TroS92], presented in figure 11. The left one, 9 transistor cell, consists of a traditional SRAM cell, plus a two-transistor exclusive-OR comparator and a pull-down transistor for the match-line. The right one, 10 transistor cell, on the other hand, includes two transistors in series for each bit-line, creating two NAND gate pull-down paths for the match-line. The first alternative needs only three transistors in total to pull-down the match-line when the value stored in the cell is different from the one on the bit-line, because it takes advantage of the complementary nature



9 xtors CAM Cell

10 xtors CAM Cell

Figure 11: The two layout alternatives for static CAM cells.

of the two outputs of the cell. The 10 transistor cell will need twice as wide transistors in the NAND gate pull-down paths in order to offset the series discharge path, and more to offset the additional capacitance of the match-line due to an extra contact per cell [10]. Yet, due to its symmetry, this cell may be laid out in less area. Since, we are more concerned with achieving high clock cycle period than area optimization, the 9 transistor cell will probably be used.

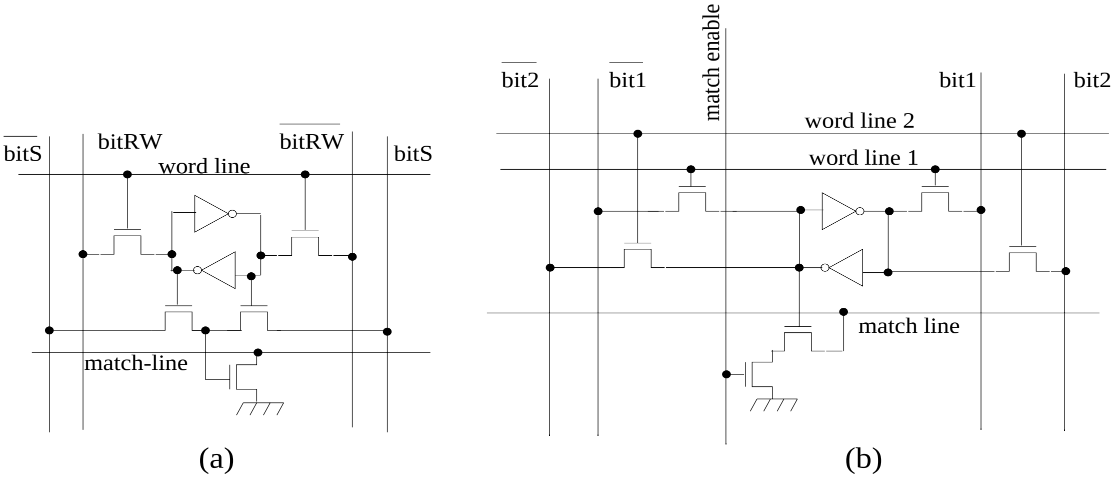


Figure 12: Content-addressable memory cells : (a) the two-ported VPout memory cell, and (b) the three-ported OutMask memory cell.

The cell for the VPout memory is shown in figure 12(a). It is a two-ported static cell. The first port (S port) is content-addressable, while the second one (W port) is a plain RAM port. The OutMask

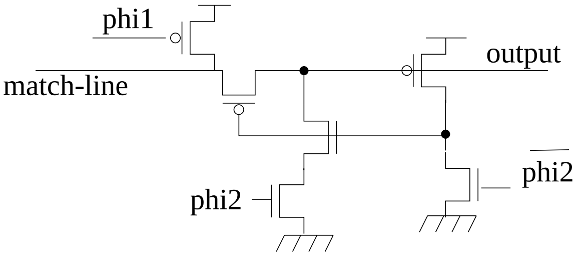


Figure 13: The match accelerator layout.

memory cell, in figure 12(b), differs in two ways. First of all it is a three-ported cell, with one CAM and two RAM ports. Furthermore, a variation of the normal CAM port is employed, for which a single bit-line (match enable) is used to search the memory only for logic one and don't care values [Sidi91]. This variation comes from the 10 transistor CAM cell, as such a 9 transistor cell variation would have an

[10] Even if a single contact is shared by both paths, the capacitance will be increased due to the extensive overlap of metal (match-line) and the n-type active area of the pull-down chain.

additional contact between diffusion and polysilicon, and therefore occupy larger area and have longer pull-down time.

There is a race condition for both cells. If a word in one of the two memories is concurrently written and searched, partial or unnecessary discharge of the match-line may occur. In order to avoid that, we can selectively discharge on every clock cycle the match-lines of the words written. Race conditions between read or write operations in the three-ported OutMask memory cannot occur, because of the way this memory is used.

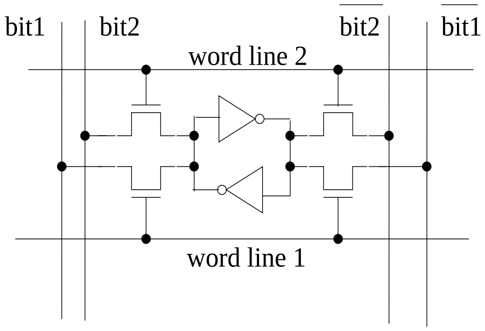


Figure 14: The two-ported SRAM cell for the CreditMask and LinkList me mories .

Since the search access of CAM memories is usually twice as slow as the read-write accesses, especially in the case where the match-line is discharged by a single memory cell, some care must be taken in order to accelerate it. The match accelerator circuit [OYT89], depicted in figure 13, could be used in order to achieve high-speed search access. This circuit detects if the match-line is being discharged. In this case, it cuts off the match-line from the output line, and the latter is discharged faster, since it has smaller stray capacitance. Yet, because such circuits are usually sensitive to noise and unstable voltage supply, and since the correctness of the search access is crucial for the operation of the credit-based flow control protocol, we believe it is safer to use a plain inverter with properly raised threshold voltage in order to achieve the desired speedup [Uyem92].

9.2 Random-Access Memory Cells

The CreditMask and LinkList memories, along with the Head-Tail register file are static random-access blocks.

The memory cell for the first two, shown in figure 14, can be the same : a two-ported static RAM cell. The cell had been layout in full-custom CMOS and its size is 11.8μm x 12.25μm. Yet, the partial overlapping of cells within the two-dimensional memory array reduces the actual cell size to 10.6μm x 10.55μm. The peripheral circuits for the two memory blocks are different since the second port of the LinkList is used for write accesses, while for the CreditMask memory is used for read/modify accesses.

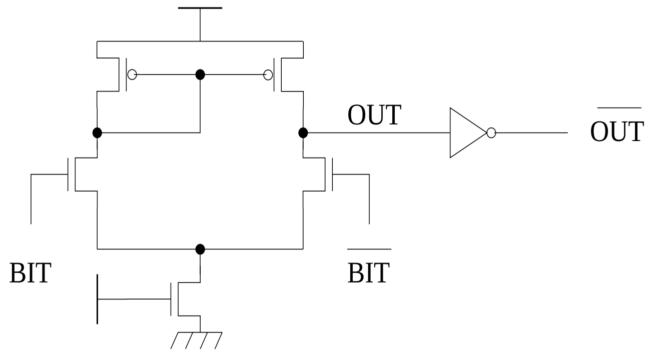


Figure 15: The single-ended operational amplifier laid out for the memories in the *QM* block.

The overall sizes of the layout of the two memories, including the bit-line drivers, sense-amplifiers and output latches, are 175μm x 1410μm and 175μm x 1402μm respectively.

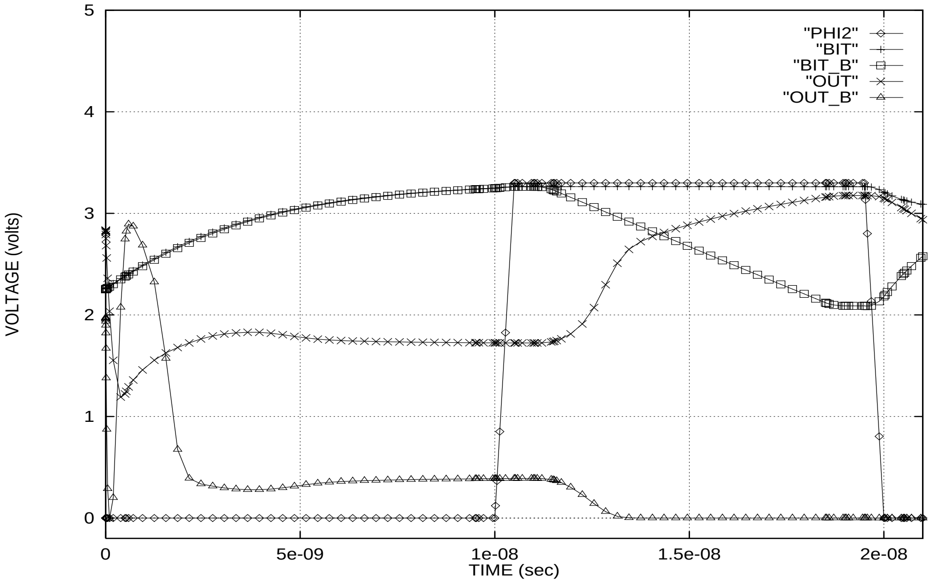


Figure 16: SPICE waveforms of the extracted netlist from the CreditMask layout, showing the operation of the sense amplifier.

The sense amplifier circuit laid out for these to memory blocks is a typical CMOS single-output operational amplifier [HaMa88], presented in figure 15. The amplifier consists of a four-transistor current mirror, connected to a current source. The output of the amplifier drives an inverter, designed to have a 1.5V threshold voltage. Its output feeds the output latch. This design was selected because it is simple, easy to layout and can safely operate under all process and environment conditions. Its only disadvantage is the delay in amplifying. Yet, the long clock cycle period (20ns) of the *QM* block is enough for the circuit to properly work. Figure 16 presents the SPICE waveforms of the extracted circuit from the CreditMask memory layout (including all parasitic capacitances). In this figure, one can see the behavior of the bit lines and the sense amplifier while reading an one. During the read phase (PHI2 high) the voltage difference between the two bit lines is almost 1V. Nodes OUT and $\overline{OUT}$ have full voltage swing and, by the end of the read phase, are at 3.3V and 0V respectively. The size of the sense amplifier cell is 10.6μm x 14.5μm. The same sense amplifier will be used with the other three memories as well.

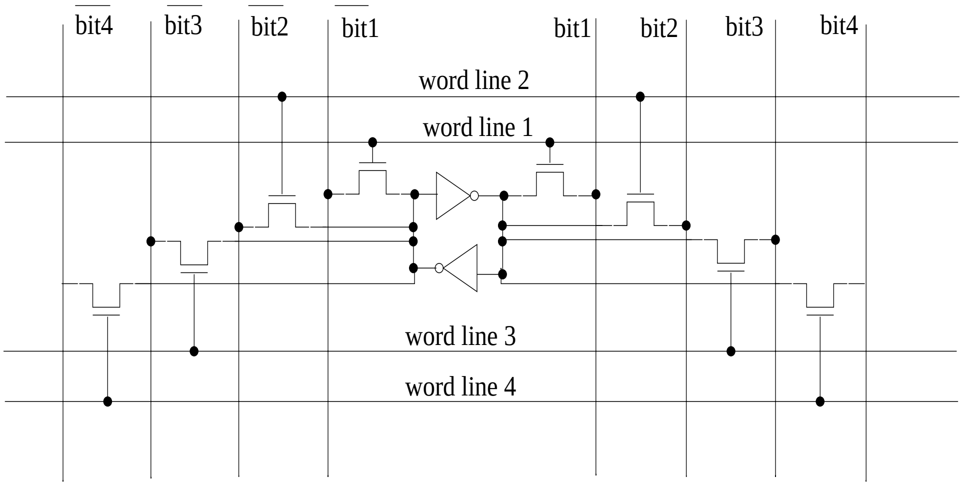


Figure 17: The four-ported SRAM cell for the Head-Tail Pointer register file.

The cell for the HTRF will be a four-ported one (figure 17), and probably the most difficult to design. Although a four ported RAM sounds extremely difficult to design, we believe it is feasible. The HTRF is not larger than a usual register file found in modern processors. These register files are multiported as well. In addition to that, the relative slow clock period aimed (20ns), supports the conception that a 54x17 bits four-ported RAM is possible. Still, during the design of this memory block, we will have to deal with the problem of fitting two rows of sense amplifiers below it (the same problem exists for the OutMask memory two).

10. The Priority Enforcer Circuit

The Priority Enforcer (PE) was laid out in full-custom CMOS first out of all the other circuits in the *Queue Management* block to be designed with full-custom VLSI techniques. The main reasons for that were: a) the increased complexity and difficulty of its design, b) its critical role in the operation of the credit-based flow-control protocol and c) the fact that there has been limited work and experience in the design of such circuits in the past.

In this section, we present in detail the operation of the Priority Enforcer circuit, analyze the most important design techniques for improving its performance and describe our implementation and its layout to be used with the *QM* block. Finally, we present two methods for designing cyclic priority enforcers.

10.1 The Operation of the Priority Enforcer

The role of the Priority Enforcer is to select one of the words in a content- addressable memory (CAM) that matched during a search operation. The inputs of the circuit are the match-lines of the CAM, i.e. a large sequence of bits containing many ones and only a few zeros, which indicate those memory words that matched with the search pattern. Its output is a sequence of equal size, where a single zero exists, the one that corresponds to the "first" one in the initial input vector. The Priority Enforcer is necessary in any application of CAMs, where multiple words may match during a single search operation, and can be used in order to implement in hardware selection algorithms such as First Come-First Served (FCFS) and Round-Robin.

In order to evaluate a certain bit of the output of the PE, we must first calculate the outputs corresponding to all the least significant bits in the input vector. To be more specific, we need to know whether one or more zeros exist in those bits or not. In the first case, the output bit is set, while in the second one it is the same with the corresponding input bit. The signal indicating the existence or not of a zero in the least significant bits is called Nobody-Else-Higher (NEH) and such a signal has to be calculated for each input bit. The equations for the output and NEH vector are : $OUT_i = \overline{NEH_{i-1}} + IN_i$ and $NEH_i = IN_i * NEH_{i-1}$ ($NEH_0 = 1$). Table 6, presents an example of the operation of the PE, that detects the leftmost zero in a 16-bit input vector.

From the above paragraph it is obvious that the problem of enforcing priority is directly proportional to the one of carry calculation and propagation in binary addition. In that case, the input carry for each bitwise addition depends on the addition result, and therefore the carry, on the least significant bits. Furthermore, an adder could be used to construct a PE that detects the rightmost zero. This is accomplished by adding one to the input vector and then using an inverter and a NAND gate per bit to calculate the final output, as explained figure 18.

IN	1	1	1	1	0	0	1	1	0	1	1	0	1	1	1	1
NEH	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
OUT	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1

Table 6: The operation of a Priority Enforcer detecting the leftmost zero in 16-bit vectors.

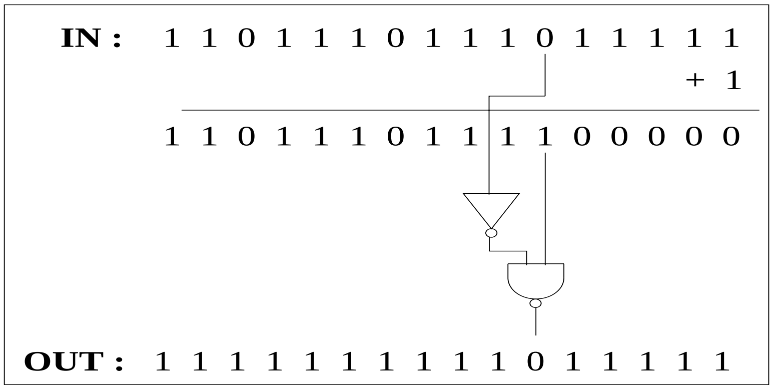

IN : 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1
+ 1
1 1 0 1 1 1 0 1 1 1 1 0 0 0 0 0
OUT : 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1

Figure 18: Example of the detection of the rightmost zero by using an adder.

The priority enforcer can also be related to the design of OR gates with large input sets. Supposing we could calculate the signal OR_i for each input bit, where $OR_i = IN_0 + IN_1 + ... + IN_{i-1}$, the output for a PE that detects the leftmost one in the input vector can be produced by using an inverter and an AND gate per bit (figure 19).

10.2 Design Alternatives for the Priority Enforcer

The Priority Enforcer can be easily designed as a regular structure with a ripple-signal, as presented in figure 20. The NEH_i calculation propagates from the top to the bottom. Taking into account that an AND gate is actually designed in CMOS as a NAND gate followed by an inverter, there is a two gates delay per bit. Thus, the total delay of the PE is 2N gates (for N inputs). This can be reduced to half by combining couples of ripple cells and modifying the second one in order to use the $\overline{NEH_{i-1}}$ signal instead, as shown in figure 21. Still the delay of N gates, restricts the use of such circuits to applications with small N (16 or 32 the most).

A PLA could also be used for the design of a PE with only two gates delay. Yet, for a large N, the PLA would be a huge one and the delay would be equal to that of two N-input gates!

In order to speed up the operation of the PE, we can take advantage of the similarity of the NEH signal calculation to the carry propagation problem, and use techniques similar to carry lookahead and

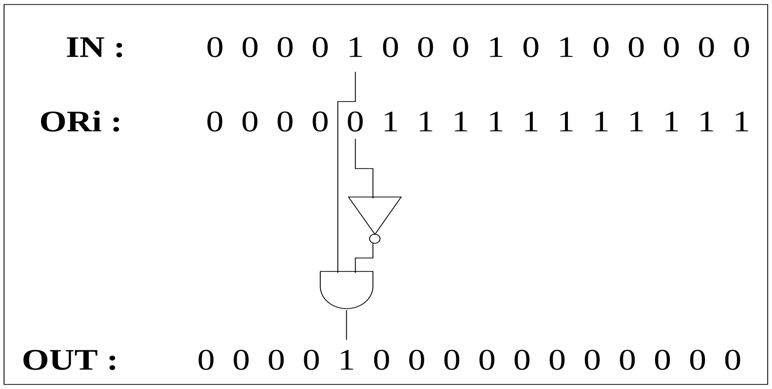

IN : 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0
ORi : 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
OUT : 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

Figure 19: Example of the detection of the leftmost one by using multiple OR gates.

carry prediction [WeEs93]. The NEH vector can be calculated by the dual tree structure presented in figure 22. The role of the upper tree is to reduce the number of input signals to a ripple structure, so that its fast evaluation is feasible. At each level, the inputs are combined in groups and for each one of them a NOR gate is used to detect if one or more zeros exist within it. The initial inputs of the tree are inverted and the outputs of the NOR gates at each level are fed (inverted too) as inputs to the next one. For example, if inputs are always combined in groups of eight, outputs of the first level indicate the existence of a zero in every group of eight inputs, those of the second one indicate the existence of a zero in every group of sixty-four inputs and so on. After an appropriate number of levels, the inputs are reduced to a small number, e.g. 8 or 16, and can be fed to a ripple-structure. This circuit calculates the NEH signal for each group formed in the last level of the upper tree. The NEH for the first group is (naturally) hardwired one.

The lower tree of the structure uses the outputs of the abovementioned ripple chain, as well as the outputs from each level of the upper tree, in order to decompose the groups, calculate the NEH signal for each one of them and, finally, evaluate the NEH signal for each original input. Each level consists of a ripple structure per group. For example, a certain level may have as inputs the NEH signals for the groups of sixty-four inputs (NEH-64) and the signals indicating the existence of a zero in every group of eight inputs (ZERO-8), produced in the upper tree. Each NEH-64 signal is used as the original NEH in a ripple chain, where the inputs are the ZERO-8 signals and the outputs are the NEH signals for every group of eight inputs. In this way, the lower tree produces the final NEH signals for each initial input by using the same number of levels with the upper one. After that, a single gate per bit is needed in order to calculate the final output of the PE. From the above description of the tree structure it is obvious that the original inputs, as well as the outputs from each level of the upper tree have to propagate until the corresponding level of the lower tree. The number of levels per tree depends on the number of the original inputs (N) and, naturally, the number of inputs that a fast ripple chain or a NOR gate may have in the available CMOS technology.

10.2 Design Alternatives for the Priority Enforcer

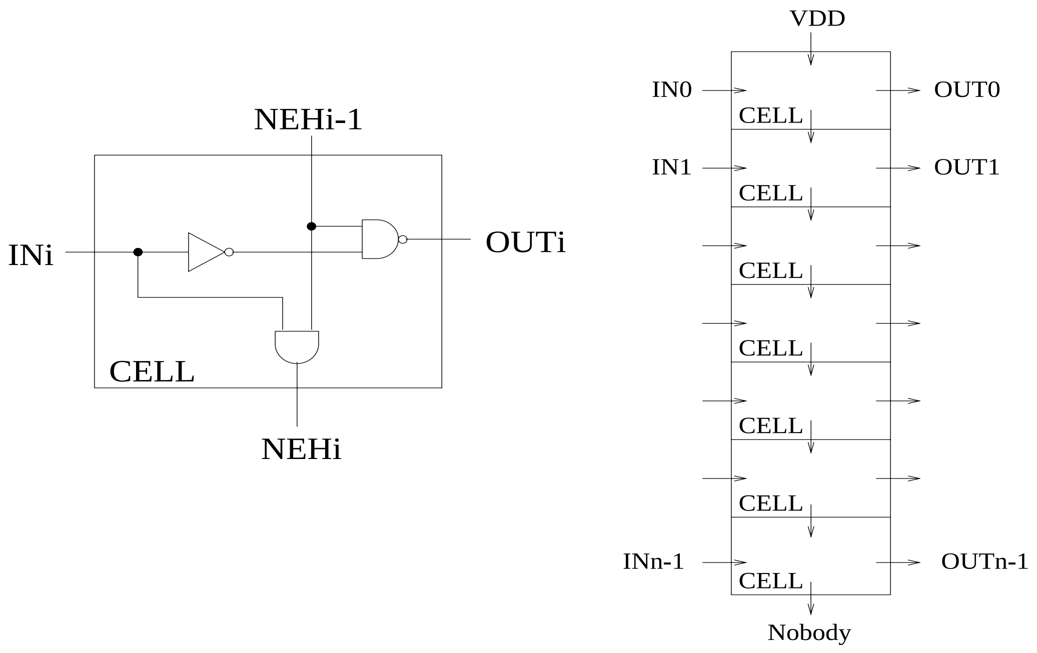


Figure 20: A ripple-signal Priority Enforcer with 2N gates delay.

10.3 VLSI Techniques for Speeding-Up the Priority Enforcer

The above described dual tree structure for the PE can be accelerated by using techniques available in full-custom CMOS design. These include dynamic circuit methods, domino timed logic and pipelining.

The upper tree in the PE consists of levels of NOR gates. Static CMOS NOR gates with large fan-in are slow, because of the pmos transistors connected in series, and occupy a large area, since they need a pmos and a nmos transistor per input. In order to avoid both negative effects we can use precharged NOR gates with domino timing [Uyem92]. A precharged NOR gate is presented in figure 23. While the PHI clock signal is low, node OUT is precharged. When PHI is set, the gate is evaluated and, if one or more inputs are high, node OUT is discharged and node $\overline{OUT}$ is set. This gate has almost the half size of the static equivalent one, since it uses a single pmos transistor for pull-up. In addition to that, multiple cascade precharged gates can be evaluated in the same clock phase. In order for a signal to be an acceptable input for a precharged gate, it must either remain low or change from low to high during the evaluation phase. The inverse transition (from high to low) is dangerous, since it would create an unnecessary and irreversible partial pull-down of the OUT node. Yet, node $\overline{OUT}$ either remains low, if no input is high, or rises from low to high during the evaluation phase, if some inputs are high, due to the pull-down of node OUT. Thus, it is safe to feed the output of such a gate to another one and evaluate them both in the same clock phase. This type of timing is called domino timing and allows us to evaluate multiple levels of the upper tree concurrently.

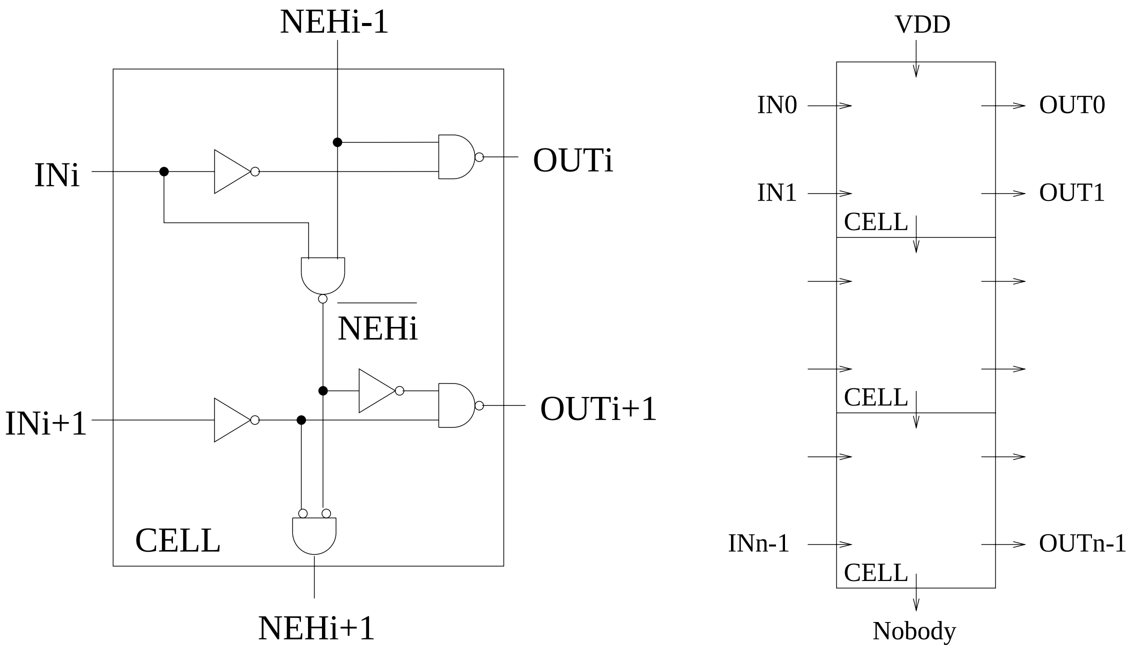


Figure 21: The modified cell for the ripple-signal PE, that reduces its delay to that of N gates.

In a similar way, we can replace the ripple structures in the lower tree with multiple precharged OR (or NOR) gates, as explained in the subsection 9.1 . The use of such gates requires all inputs of the ripple structured to be inverted, but in this case we will be able to evaluate multiple gates (i.e. multiple tree levels) in the same clock phase. The only disadvantage is that one would have to design another NOR gate (with different fan-in) for each cell in the ripple structure previously used. The first one would have a single input (inverter), the second one two and so on. In order to avoid that one can merge multiple NOR gates into a Manchester chain circuit [WeEs93], presented in figure 24. A Manchester chain is a dynamic circuit that can be used for evaluation of multiple OR-type results. During the low phase of the clock signal PHI, all nodes INT_i are precharged. When is PHI high (evaluation phase), all nodes are discharged through the series of nmos transistors, upto the point of the chain where the first low input appears. Supposing that the IN_i signals are the inverted original inputs, OUT_i is the result of a NOR gate on the first (i-1) inputs. One can also notice that the outputs follow the domino timing, thus can be fed directly to a second chain, which is evaluated in the same time with the first one. The first input of a Manchester Chain can be the NEH signal of the least significant inputs, in case it does not operate on the first group of inputs.

In case the input set is very large, the use of dynamic precharged logic with domino timing may not be enough by itself to significantly reduce the delay of the Priority Enforcer. For example, if N=512 and all the gates (NOR and Manchester chains) have 8 inputs, there are $2 * log_8 N - 1 = 5$ levels in the dual tree structure, thus 5 levels of logic to be evaluated in a sequential manner. This is a significant

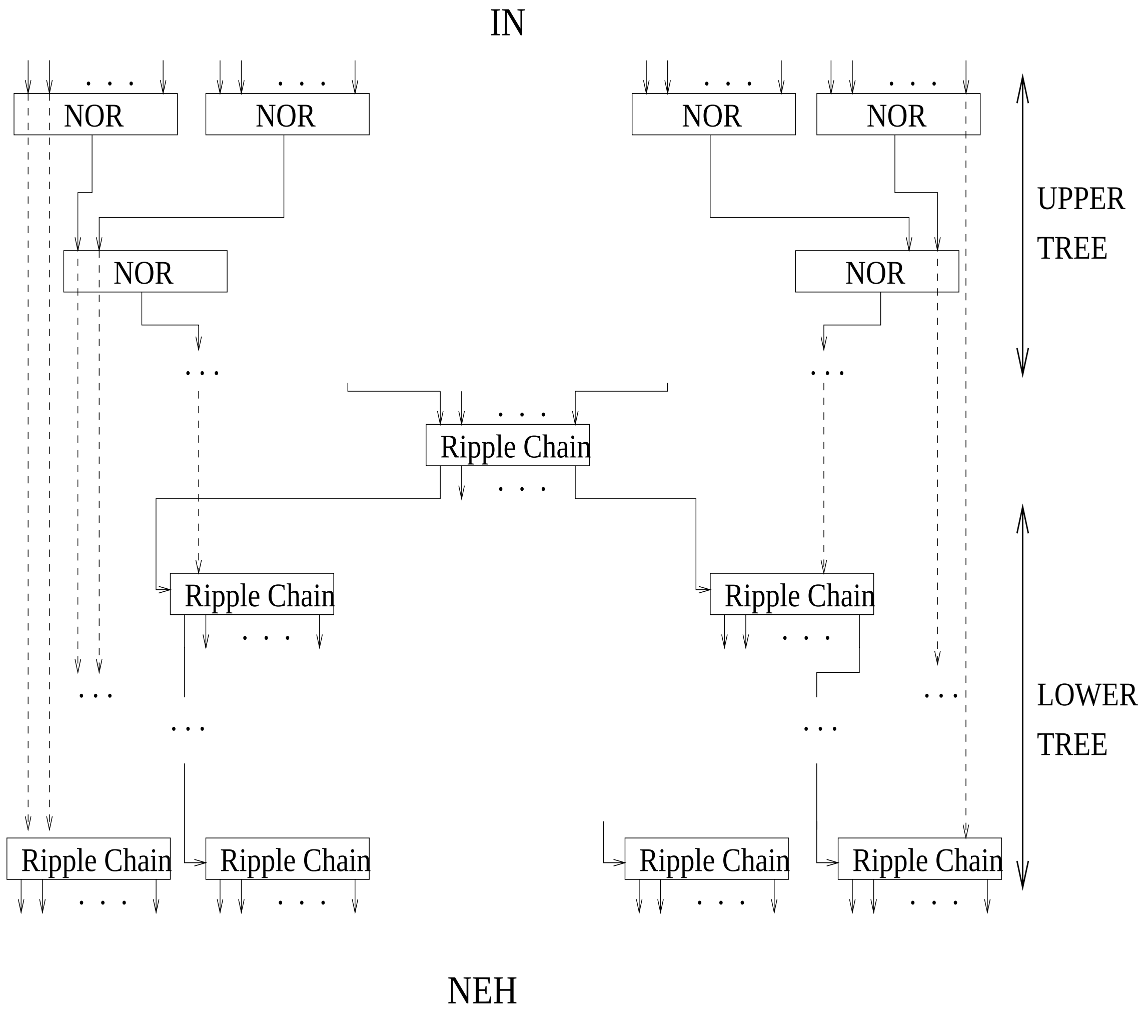


Figure 22: The dual tree structure for a high speed Priority Enforcer.

improvement compared to the N=512 levels of logic in the simple ripple structure, but may still be infeasible in designs with high clock speed, such as the *ATLAS* I switch (50 Mhz). In order to overcome this problem, we can add pipelining between the levels of the dual tree. One can separate the tree levels in pipeline stages, so that the desired clock frequency is achieved. Each stage must comprise of at least one tree level. Neighboring stages do not have to operate on the same data set in consequent clock cycles. Since precharged logic is used, stages can operate in consequent clock phases. While the one stage is evaluated, the next one is precharged and via versa. Thus two pipeline stages on each inputs set in every cycle. Still, we would like to notice that, the addition of pipelining does not reduce the overall delay of a single evaluation of the PE circuit. Yet, it raises the rate at which we can feed inputs and get results from it to one set per clock cycle, which is important for high performance designs.

10.3 VLSI Techniques for Speeding-Up the Priority Enforcer

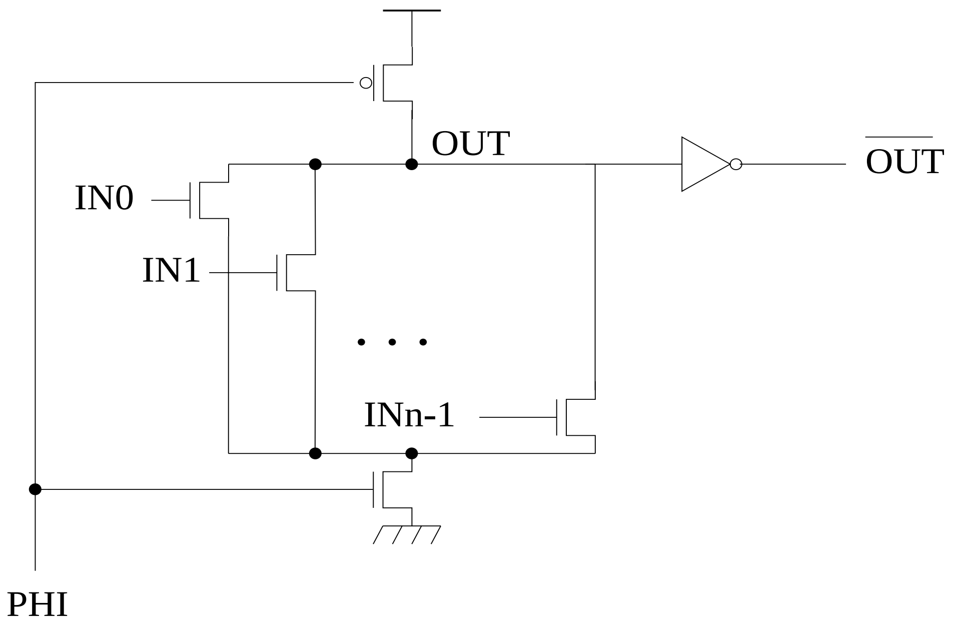


Figure 23: A N-input precharged NOR gate with an output inverter for domino timing.

Finally, a last source of delay in the dual tree structure of the PE, is the propagation of the original inputs and the results of the upper tree throughout the circuit. Folding the two trees together eliminates this delay. Corresponding levels of the two trees become neighboring and intermediate results have to propagate over a small distance. This folding is particularly useful when the result of the PE has to be used as an address for a read/write access to the same CAM, since the result comes out from the same side that the inputs came from (the side where the CAM is).

10.4 The Priority Enforcer in the Queue Management Block

The Priority Enforcer laid out in full-custom CMOS for the *Queue Management* block of the *ATLAS* I switch has 256 inputs, since it is fed with the combined match-lines from the VPout and OutMask memories (256 words each). It detects the first word that matched during the search operation, always starting from the top (word 0). The dual tree structure with the techniques described in the previous section were applied in its design, adapted to the 0.5um CMOS technology provided by SGS-Thomson. Yet, no folding of the two trees was necessary, since the output is used as a decoded address to both the CreditMask and OutMask memories. The floorplan (in a block diagram style) of the PE circuit is shown in figure 25, along with the sizes of the various parts of the circuit. The floorplan has been turned right by 90 degrees in order to be easier to examine.

Both trees in the PE consist of a single level. In the upper tree, there are 16 precharged NOR gates

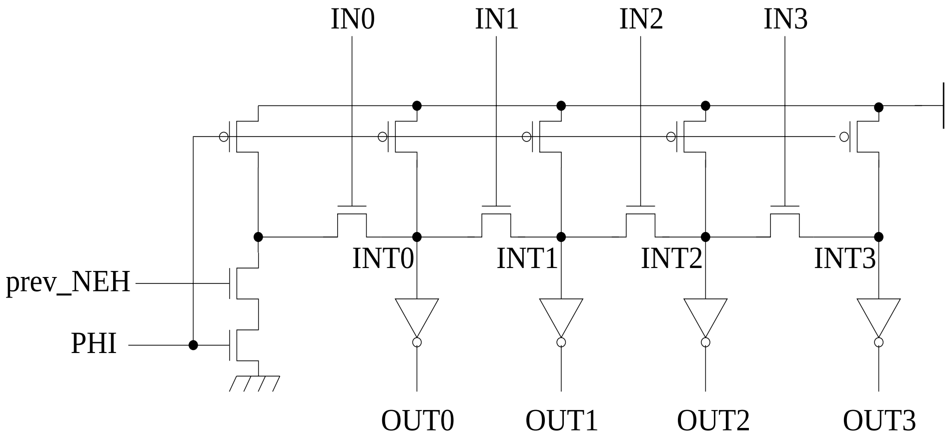


Figure 24: A 4-input Manchester Chain circuit.

with 16 inputs, which produce the signals indicating the existence of a zero in every group of 16 inputs (ZERO-16). The pull-down part of each NOR gate spreads over the vertical area of the corresponding inputs. In this way we avoid having to bring the 16 inputs close together (loss of area in turning wires), without sacrificing speed, which is proportional to that of reading from a memory of 16 words (where the pull-down paths also spread in the vertical dimension). Instead of feeding the ZERO-16 signals to a single Manchester chain, we use 16 precharged NOR gates to get the NEH signal for each group of 16 inputs. These gates achieve faster (parallel) calculation without any area cost, as the are placed in the otherwise unutilized area at the top and bottom of the single Manchester chain. The first gate has a single input (inverter), the second one has two and so on. The output of the last gate indicates whether a single match exists or not. The output of each NOR gate is used as a previous NEH input signal signal to the corresponding Manchester chain in the lower tree. In other words, the output of the first NOR feeds the second Machester chain, the output of the second one goes to the third Machester chain, etc.

The lower tree consists of a level of 16 Manchester chains with 16 inputs. Each chain has been broken in two parts of 8 inputs each, where the last output of the first one is used as a NEH input to the second one, as presented in figure 26 (the precharge pmos transistors have been omitted). The connection of the two sub-chain in this way is possible because of the domino timing of their inputs and outputs. In order to further reduce the delay of the chain, the width of the nmos transistors is increased as we move from the right to the left side of each chain. In this way, the current that can pass through the nmos transistors in the chain constantly increases as the charge flows from the intermediate nodes to the ground, and thus, all nodes in the chain are pulled-down faster. In figure 27, there are the waveforms from the evaluation of a Manchester chain with all inputs high (worst case), as produced by SPICE simulation on the netlist extracted from actual layout, including all the parasitics capacitances. One can see that, when the PHI clock signal is set (evaluation phase), the OUT7 node is pulled-up, causing the nodes in the second sub-chain to be evaluated as well. Within 3ns, the last output (OUT15) is pulled-up

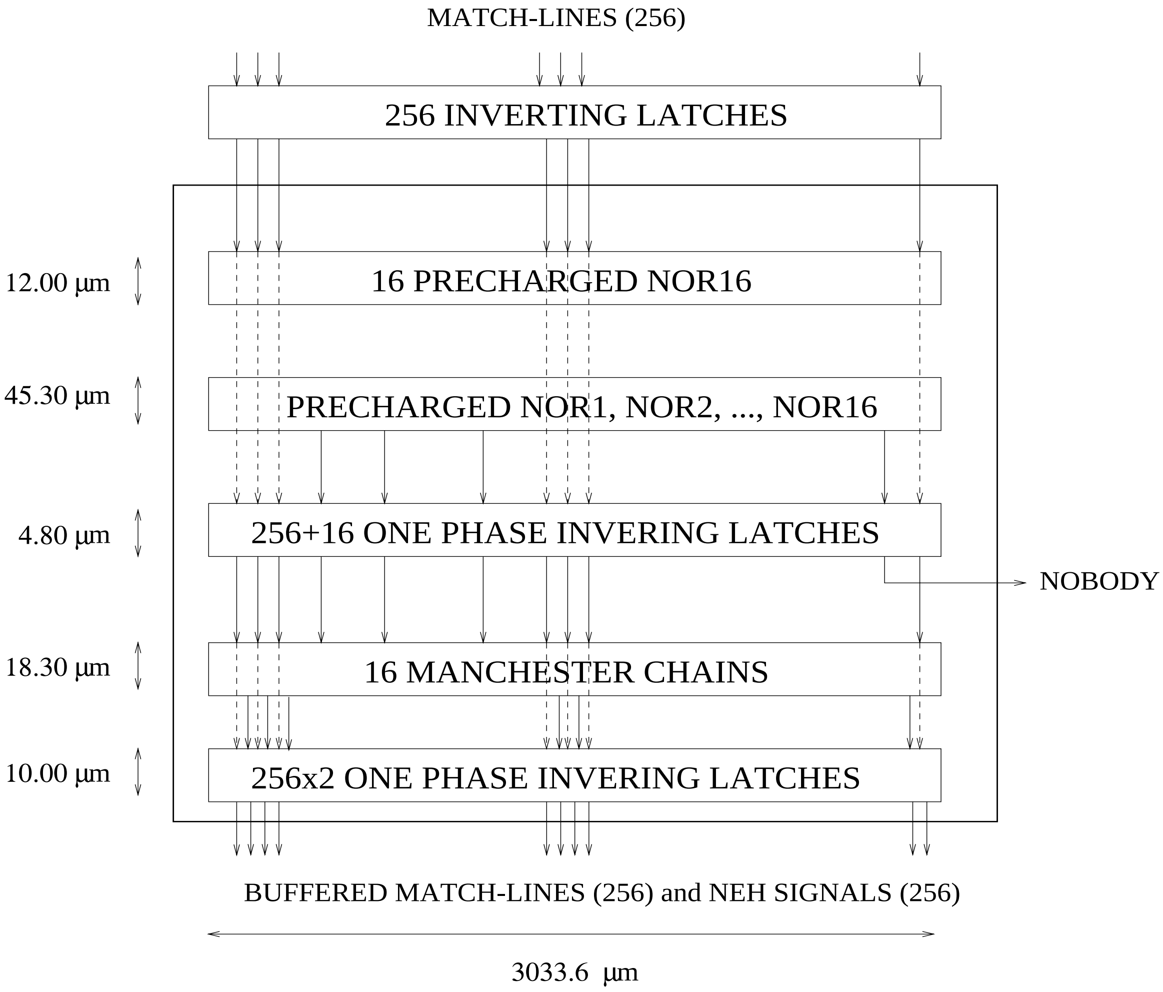

MATCH-LINES (256)
256 INVERTING LATCHES
12.00 µm
16 PRECHARGED NOR16
45.30 µm
PRECHARGED NOR1, NOR2, ..., NOR16
4.80 µm
256+16 ONE PHASE INVERING LATCHES
NOBODY
18.30 µm
16 MANCHESTER CHAINS
10.00 µm
256x2 ONE PHASE INVERING LATCHES
BUFFERED MATCH-LINES (256) and NEH SIGNALS (256)
3033.6 µm

Figure 25: The floorplan of the Priority Enforcer in the Queue Management block.

as well and the evaluation of the Manchester chain finishes.

The circuit is separated in two pipeline stages by a column of one phase pipeline latches. Thus, it takes one clock cycle (20ns), or two clock phases to produce a single result. The fist stage includes the upper tree, as well as the NOR gates used instead of the intermediate Manchester chain. The second stage includes only the lower tree. The gates needed in order to evaluate the final output from the initial inputs and the NEH signal where not added, since their functionality will be included in the wordline driver of the CreditMask memory. Hence, the outputs of the circuit are the NEH vector and the initial inputs (matched lines) interleaved.

The total size of the Priority Enforcer circuit is 3033.6μm x 90.4μm, including the output latches. The horizontal dimension could be further reduced, but we choose not to in order to guarantee the correct operation under all circumstances. The vertical dimension (11.85μm x 256 $=$ 3033.6μm) was fixed from

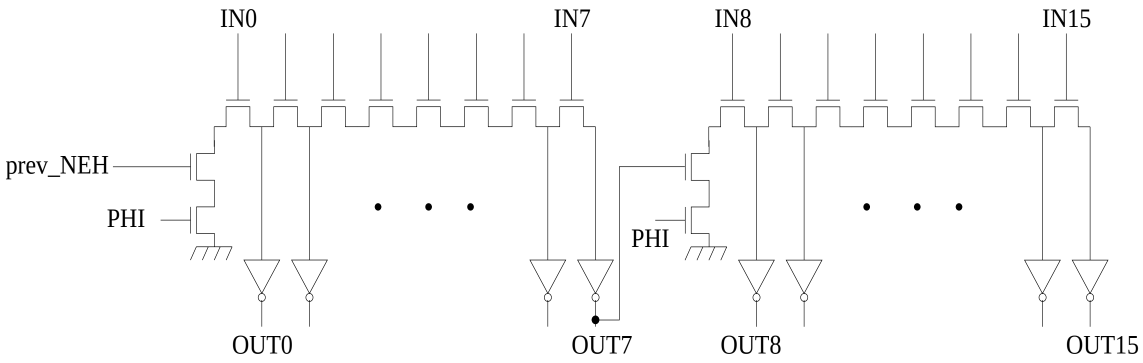


Figure 26: The 16-input Manchester Chain circuit used in the Queue Management block.

the start, since the PE circuit must match with that of the OutMask CAM. The circuit was tested with the STSPICE transistor-level simulator [ST96], provided by SGS-Thomson, under all possible process and environment conditions.

10.5 Cyclic Priority Enforcers

As mentioned in section 9, the Priority Enforcer may have to be a cyclic one, in order to guarantee randomness and fairness in the distribution of incoming credits in the case of merging flow-groups. A cyclic PE does not search for the first word that matched always from a static point (top or bottom). The starting point moves cyclically, so that all zeros in the input vector have an equal probability to be selected.

There are two possible ways to implement the cyclic motion of the starting point. The first one is to start searching from one place below of the previously selected word. If word 255 was previously selected, we start from the top. The second one is to move the starting point cyclically one place at the time : first word 0, then word 1 , ..., etc. Both methods have advantages and disadvantages and may prove to be the appropriate one to use.

Building a cyclic PE does not demand the design of a completely new circuit. We can built cyclic Priority Enforcers of both types by using two simple (or static) PEs, like the one described in the previous subsection. At first, we examine the cyclic PE where the starting point is always one place below from the previously selected word. The first static PE always operates on the original inputs, i.e. the match-lines. The second one operates on the initial inputs, after they have passed through OR gates with the NEH produced in the previous cycle. In this way, we set all zeros above the place selected in the previous cycle (including that one). If the second static PE detects a zero, we keep as final NEH vector the one it produced. This indicates the first zero existing below previously selected word and until the bottom (word 255). If not, we use the NEH vector of the first static PE, which identifies the first zero from the

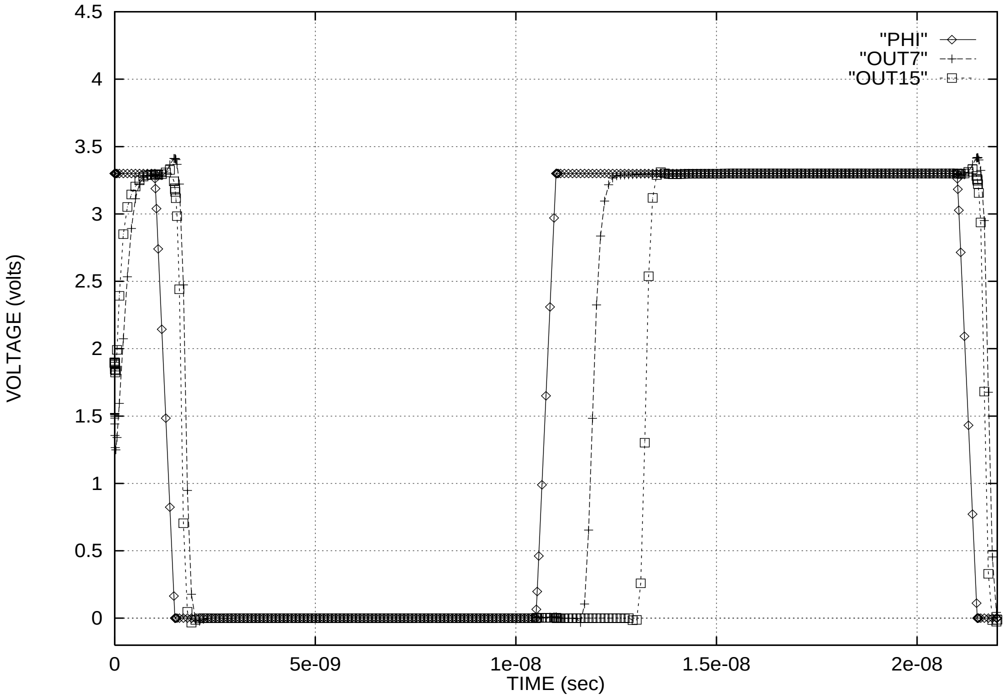


Figure 27: SPICE waveforms of the extracted netlist from the 16-input Manchester chain layout.

top and, therefore from the top until the previously selected word. Thus, by using the signals indicating whether a static PE found a zero or not (signal nobody), we can implement the cyclic motion, at the cost of two static PE, a buffer for the previous NEH vector and a multiplexor.

The cyclic Priority Enforcer for the second method can be constructed in exactly the same way, by replacing the buffer for the NEH vector with a "cyclic" shift register. This register would consist of 256 simple latches connected in a row. On every clock cycle, the contents of the latches are shifted by one place. Initially all latches have an one stored. On each clock cycle a zero is inserted in the top latch, apart from the case when the pattern 01 was stored in the last two latches. In this case, all latches are reset. With this "cyclic" shift register we create the NEH vector as if the selected word moved cyclically by one place on every clock cycle. By using both static PE in the same with as with the first method, we have a cyclic Priority Enforcer where the starting point cyclically shifts one place on each cycle.

11. Conclusions

Throughout this work, it is obvious that maintaining high performance data structures for cells in ATM switches, is far more complicated than it was in older switches. Additional queues have to be implemented due to the priority-based routing and the multicasting support, while the rate of events has risen due to the high link throughput and the flow control protocol needed. The demands in both speed and rate of operations supported, can be met by using techniques such as pipelined and superscalar processing of events. This approach significantly reduces hardware complexity, especially in the control unit, and enables higher utilization of hardware resources, such as memory ports. Yet, one should also consider the hazard situations that may arise, and provide for a solution through bypassing datapaths.

The *Queue Management* block, that was presented, uses this approach to maintain 54 output queues and one pool of cells in the shared data buffer of the *ATLAS* I switch. It contains five memory blocks, whose size depends on the buffer's size, and the number of ports for each one is restricted to minimum. The block also implements the proper bypass rules for safe operation and supports management commands. It was functionally simulated, with a clock cycle precisions, in order to ensure correctness of its operation prior to the VLSI implementation. In addition, the the Priority Enforcer circuit was both studied and designed, while the full-custom design of the memory cells necessary was analyzed.

The *Queue Management* block, as well as the whole *ATLAS* I switch, is currently in the VLSI design phase. After that, the testing and post-layout verification phase will follow. The switch is expected to be sent for fabrication in March 1997. Apart from the usual post-fabrication testing of the chip, its operation will also be presented by using the ASICCOM demonstrator and test/management software developed within the same project.

Acknowledgments

ATLAS I is being developed within the ASICCOM project, funded by the European Union ACTS Programme.

Many people involved in the ASSICOM project have contributed to this work. I wish to acknowledge in particular Peny Vatsolaki for her contribution to the architecture of the *Queue Management* block, Chara Xanthaki for her help with implementation issues and, finally, professor Manolis Katevenis for his overall guidance.

I also want to thank my parents for their love and support.

References

[CoST88] J. Coudreuse, W. Sincoskie, J.S. Turner: “Guest Editorial in Broadband Packet Communications”, *IEEE Journal on Selected Areas in Communications*, vol. 6(8), December 1988, pp. 1452-1454.

[Gros92] K. Grosspietsch: “Associative Processors and Memories: A survey”, *IEEE Micro*, June 1992, pp. 12-19.

[HaMa88] M. Haskard, I. May: “Analog VLSI Design, nMOS and CMOS”, Prentice Hall, ISBN 0-13-032640-2, 1988.

[HlKa88] M. Hluchyj, M. Karol: “Queueing in High-Performance Packet Switching”, *IEEE Journal on Sel. Areas in Communications*, vol. 6, no. 9, December 1988, pp. 1587-1597.

[KaSS96] M. Katevenis, D. Serpanos, E. Spyridakis: “Credit-Flow-Controlled ATM versus Wormohole Routing”, Technical Report FORTH-ICS/TR-171, "ICS, FORTH, Heraklio, Crete, Greece, July 1996. URL: file://ftp.ics.forth.gr/tech-reports/1996/1996.TR171.ATM_vs_Wormhole.ps.gz

[KaSV96] M. Katevenis, D. Serpanos, P Vatsolaki: “ATLAS I: A General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control”, Hot Interconnects IV Symposium, Stanford Univ., CA, USA, Aug. 1996. URL: file://ftp.ics.forth.gr/tech-reports/1996/1996.HOTI.ATLAS_I_ATMswitchChip.ps.gz

[KaVE95] M. Katevenis, P. Vatsolaki, A. Efthymiou: “Pipelined Memory Shared Buffer for VLSI Switches”, *Proceedings of the ACM SIGCOMM’95 Conference*, Cambridge, Ma., USA, 30 August - 1 Sep. 1995, pp.39-48. URL: file://ftp.ics.forth.gr/tech-reports/1995/1995.SIGCOMM95.PipeMemoryShBuf.ps.gz

[KSVMC96] M. Katevenis, D. Serpanos, P Vatsolaki, E. Markatos, K. Courcoubetis: “ATLAS I Architecture: Architecture of the ATM Switch Chip of ASICCOM”, version 1.1, ASICCOM Consortium Internal Document, March 1996.

[LeBo92] J. LeBoudec, “The Asynchronous Transfer Mode : A Tutorial”, *Computer Networks and ISDN Systems*, vol. 24, no. 4, May 1992.

[OYT89] T. Ogura, J. Yamada, S. Yamada, M. Tan-No: “A 20-kbit Associative Memory LSI for Artificial Intelligence Machine”, *IEEE Journal of Solid-State Circuits*, vol. 24, no. 4, August 1989, pp. 1014-1020.

[PaHe95] D. Patterson, J. Hennessy: “*Computer Architecture : a quantitative approach*”, Morgan Kaufman Publishers, ISBN 1-55860-329-8, 1995.

[PaHe93] D. Patterson, J. Hennessy: *“Computer Organization : the hardwaresofware interface*”, Morgan Kaufman Publishers, ISBN 1-55860-281-X, 1993.

[Sidi91] S. Sidiropoulos: “Fast Packet Switches for Asynchronous Transfer Mode”, Technical Report FORTH-ICS/TR-25, *ICS, FORTH, Heraklio, Crete, GR*, August 1991, 69 pages.

[ST96] “ST-SPICE User Manual”, SGS-Thomson, July 1996.

[Syn94] “HDL Compiler for Verilog Reference Manual”, Synopsys Inc, March 1994.

[TaFr88] Y. Tamir, G. Frazier: “High-Performance Multi-Queue Buffers for VLSI Communication Switches”, *Proc. of the 15th Int. Symp. on Computer Architecture*, ACM SIGARCH vol. 16, no. 2, May 1988, pp. 343-354.

[Toba90] F.A. Tobagi: “Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks”, *Proceedings of the IEEE*, vol. 78, January 1990, pp. 133-167.

[TroS92] N. Troullinos, C. Stormon: “Design Issues in Static Content-Addressable Memory Cells”, CASE Center Technical Report No. 9208, *CASE Center, Syracuse University*, August 1992.

[Uyem92] John P. Uyemura: *“Circuit design for CMOS VLSI*”, Kluwer Academic Publishers, ISBN 0-7923-9184-5, 1992.

[Veri94] “Verilog-XL Reference Manual”, Cadence Design Systems Inc., v. 2.1, Decmber 1994.

[WeEs93] N. Weste, K. Eshraghian: *“Principles of CMOS VLSI Design — a Systems Perspective*”, second edition, Addison-Wesley, ISBN 0-201-53376-6, 1993.