A Scalable Queue Management for Shared Buffer Switch Architecture

Pi-Chung Wang , Chia-Tai Chan and Yaw-Chung Chen

Abstract— **In this paper, we present the design of a scalable queue management for shared buffer switch architecture by using non-linked-list implementation. In the traditional switch design, linked-list is the widely adopted implementation to maintain the packet buffer. However, to design a linked-list architecture in hardware increases the size of chip and also limits the operating frequency. By using the non-linked-list design, the proposed architecture not only offers a wire-speed transmission rate, but also provides an efficient Shifting Page Queue Manager (SPQM) for queue management. It may fulfill the emerging needs for high-speed LAN switches. Moreover, it is feasible in implementation.**

Index Terms— **Ethernet, Gigabit Networking, VLSI.**

I. Introduction

WITH the demand towards higher-speed and high bandwidth, the LAN switches become popular and acceptable on today's LAN systems. Among existing switching architectures, the shared buffer switch is favorable for its efficient buffer utilization. In the traditional switch design, linked-list is the widely adopted implementation to maintain the packet buffer. However, to design a linked-list architecture in hardware increases the size of chip and also limits the operating frequency. The main concerns of shared buffer approach are the need for memory high-access-rate and the increased complexity of the queue manager if it were to support multicasting and multi-priority classes. By adopting the newest RAM technology such as DDR-RAM or Rambus, memory speed is fast enough to support several Giga bps ports. Obviously, to design an effective queue manager is necessary for high speed LAN switches. In this paper, we propose a scalable queue management for shared buffer switch architecture by using non-linked-list implementation. The proposed architecture not only offers a wire-speed transmission rate, but also provides an efficient Shifting Page Queue Manager (SPQM) for queue management. It may fulfill the emerging needs for high-speed LAN switches.

The organization of this paper is as follows. Section II makes an overview of the shared buffer queue management and switch architecture. In Section III, we present the architecture and operation of our proposed buffer management. Section IV contains a thorough description of the proposed SPQM. Finally, Section V gives some concluding remarks.

The authors are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. (e-mail: {pcwang, ycchen}@csie.nctu.edu.tw).

Chia-Tai Chan is now with the Telecommunication Laboratories, Chunghwa Telecom Co., Ltd, Taipei, Taiwan,R.O.C.(e-mail: ctchan@ms.chttl.com.tw).

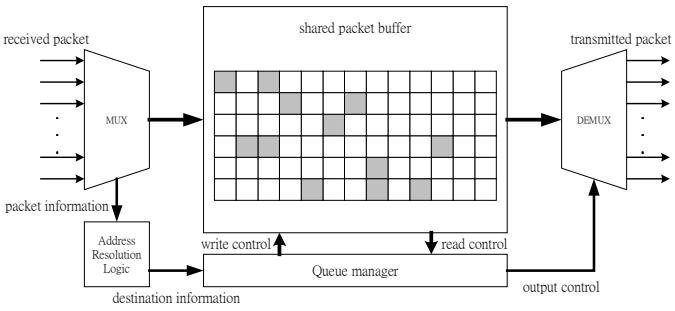

Fig. 1. The shared buffer switch architecture overview.

II. Overview of the Shared Buffer Switch Architecture

In the shared buffer switches, packets from all input ports are stored in buffer memory. To maintain correct delivery, packets are stored into associated output queues according to their destination ports by the queue manager, as shown in the Figure 1. The queue manager generates control signals which control the read/write accesses of the packet buffer and the output demultiplexer to achieve packet switching.

Various implementations of the queue manager have been proposed with the shared buffer switches. The address-based approach has been widely adopted in most existing designs including [2][5][6]. Each packet in the shared buffer has a corresponding next address field which designates where the next packet of the same output queue is stored. These next address fields form linked-lists of the output queues. Packets are read from the heads of the lists for transmission in the output ports, and the incoming packets are attached to the tails of their associated linked-lists. However, the existing linked-list implementation of the address-based queue management is inefficiency. First, it reduces the utilization of buffer space by using the extra address fields to provide FIFO service. Second, some buffer memory locations, named bubbles, have to be reserved for linked-list operations. These bubbles also reduce the buffer capacity.

Since the packet size is variable, such as the packet length of Ethernet frame might vary from 64 bytes to 1518 bytes. To utilize the packet memory space well, current design handles packet memory allocation by segmenting the whole packet memory into equal size block (page here after). Once receive a packet, buffer manager would check its size and allocate corresponding minimum number of pages. This scheme avoids the external fragmentation problem, but causes the internal fragmentation problem. To choose

a appropriate page size might alleviate this. Clearly, it needs extra descriptor because one packet may be divided into several pages. Hence, the smaller the page size is, the more descriptor it needs.

III. Proposed Shared Buffer Switch Architecture

In the proposed switch architecture, packet memory is shared dynamically among the active ports, as shown in Figure 2. Once a packet arrives the switch input port, it will be transmitted to PMC for further processing. The PMC will check the packet header and modify it if necessary. PMC also allocates the available page to store the incoming packet. The available page information is stored in a word vector. Each page map to an element in this vector as show in Figure 3. The word vector consists of three field: valid bit, used bytes and reference count. While a page is used, its corresponding valid bit will be set to 1. PMC have to check the packet size and find out enough unused pages to store the packet. If the page size is 256 bytes, at most 6 pages is needed for the maximum Ethernet frame size. The field "used bytes" keeps the information about how much useful data is stored in this page. PMC updates this field after laying up the packet into packet buffer.

After storing the packet, PMC transmits the packet header and the packet storage information to the forwarding engine. If the packet is a multicast/broadcast packet, forwarding engine will update the reference count in PMC. The reference count records the number of ports to which this packet will be send. It is used for multicast/broadcast capability. If the reference count is 1, it means this packet is unicast. While this packet is transmitted, the page will be released at once. Otherwise, the reference count will be subtracted by 1 and its related page will be released until the reference count equals 0. The reference count should be less than the total number of ports in a switch.

While the queue manager notifies PMC to send a packet, PMC will receive page addresses belonged to the same packet and calculate the packet length by accumulating the "used bytes" field. Then, it sends request to the packet buffer for transmitting exactly amount of data to the output port. The size of the word vector is proportional to the buffer size. For example, a packet buffer with 2 Mbytes

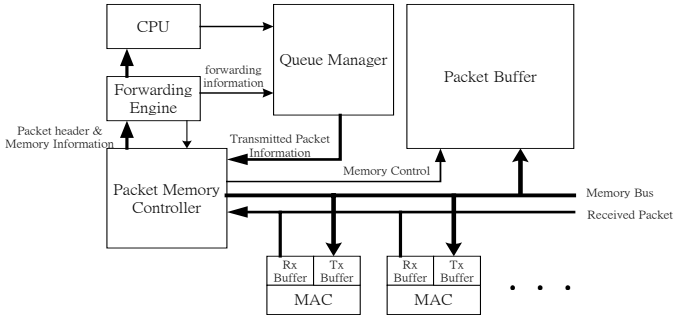

Fig. 2. The proposed shared buffer switch architecture.

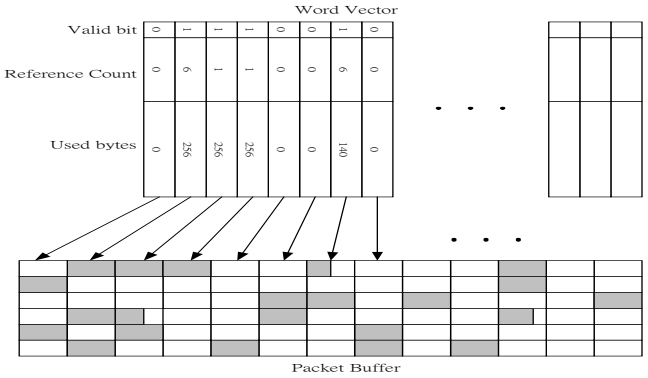

Fig. 3. The relation between the word vector and packet buffer.

will be divided into 8K pages with 256 bytes per page. The length of "unused bytes" is 8 bits, and 7 bits is large enough to keep the reference count. Thus 2 bytes are needed for each page and the total size of word vector is 16 Kbytes. In addition, the PMC can be extended to support multi-shared buffer by adding per-buffer word vector and extra load-balance control logic.

IV. A Scalable Buffer Manager

To design a scalable queue manager, the implementation architecture must has a regular structure that well suits to the VLSI design and the number of flows can grow flexibly. The logic diagram of queue manager is shown in Figure 4. Each FIFO queue is with associated SPQM for an output port. Assume that the ith Packet is forwarded to the output port 1, and the value in the parentheses is the address of the related page. As shown in the diagram, the i_{th} packet is divided into pages 2, 5 and 6 respectively. Each queue element should carry information of page physical address (PA here after) and the location of this page in a packet. It is essential to perform packet level transmission. Therefore, the queue header will fetch the page address until the last page of the packet. The last page of the packet can be identified by the *last page flag*.

Therefore, if we use a 16-bits register to store the page information, we can set the first bit as *last page flag* and still have 15 bits which can allocate up to 8 MB packet buffer with 256 bytes per page. This is large enough in a LAN environment. By building per-port FIFO, we can reduce the complexity of hardware implementation of linked-

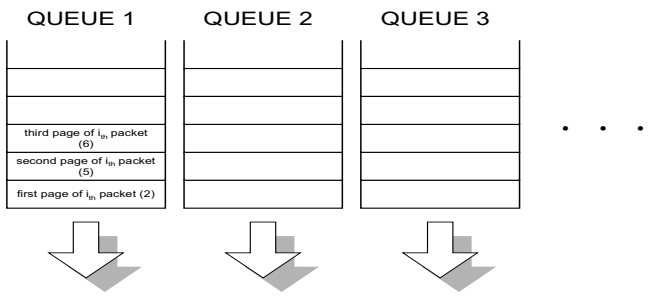

Fig. 4. The logical diagram of queue manager.

list. One may argue that the utilization of queue element might become the system bottleneck. We will present two enhancements architecture in Section IV.A to solve this problem.

A. Realization of the SPQM

We proposed a feasible architecture to realize the SPQM which is similar in idea to the available VLSI Sequencer chips [3] with additional circuit added, as shown in Figure 5. The service discipline of the architecture is explained in the following. Each module in the SPQM includes one register, multiplexer, valid bit and a simple control logic. The PA is stored in the 16-bit register. The valid bit of the empty module is set to 1. Otherwise, it is set to 0. When the new PA is to be inserted into FIFO, the controller just appends the new PA to the empty modules in the FIFO. By performing the logical AND at each module with its valid bit and the inverted valid bit in the previous module, only the first empty module will produce 1. Thus the control logic can decide whether the module should accept the PA or not. Besides, the behavior in Module 1 is a little bit different with others since it is the first module in the FIFO. It only has to check its valid bit for verifying if it is the first empty module.

When an HOL page number is scheduled for transmission, its page number is first retrieved from controller, and the content of all 16-bit registers will be shifted one position to the right by triggering the *Advance* signal. Hence the PA in the FIFO will be overwritten automatically. The head module will check if the fetched PA is the last page of the packet by checking the first bit of register, which is *last page flag*. If the value is 0 (false), the Head Module will fetch the next one PA and repeat the above steps until the last one. The fetched PA will be stored in the address buffer of head module. Once it collects a complete packet, it will notify the PMC to transmit packet from the collected PA. Once the PMC has transmitted the packet, its related reference count in word vector will be subtracted by 1.

Obviously, this architecture is relative simpler than linked-list approach and is suitable to implement in hardware. The total demanding address storage is the size of address buffer (1518/page size) plus the address storage in all modules, which is about half storage as needed in the linked-list approach. Thus this architecture is applicable to implement in high speed.

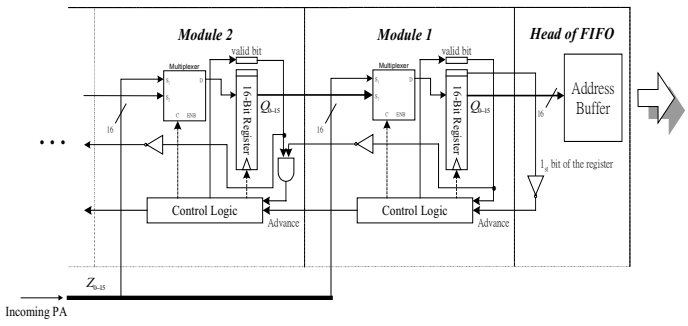

Fig. 5. The architecture of the SPQM.

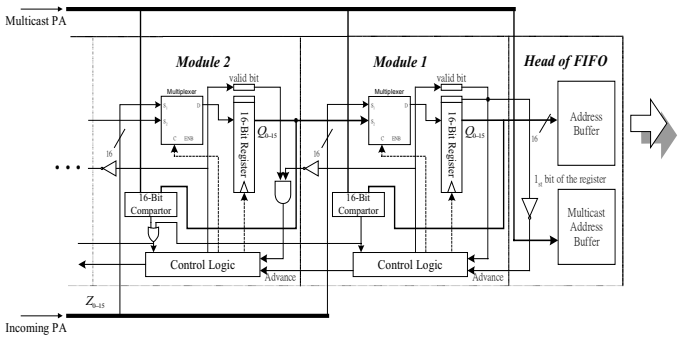

Fig. 6. The multicast architecture of the SPQM.

B. SPQM Multicast Architecture

In the previous architecture, the page's PAs of multicast/broadcast packet will be appended to several SPQMs. The PMC can only free the allocated buffer space after the packet leaving all ports, i.e. the reference count is 0. If we can transmit the multicast packet at the same time, then we can release the buffer space earlier and increase the buffer utilization.

The architecture of SPQM can be modified to transmit multicast packet to output port at the same time by adding a 16-bit compartor at each module and a multicast buffer at the head of FIFO, as shown in Figure 6. Once the head of FIFO identifies this is a multicast packet, it will broadcast its PAs to all other FIFOs. The head of FIFO which received these PAs will store the information in the extra multicast address buffer. At the same time, each module will receive these multicast PAs. By comparing the value with the register value of module, the control logic is able to decide whether to retain or to shift-right the content of the registers. The page's PAs of multicast packet in the register will be overwritten automatically. Then, the multicast packet will be transmitted from FIFOs.

Since the multicast packet will be transmitted from FIFO at almost the same time in this architecture, the related buffer space can be released earlier. Thus we can improve the buffer utilization as well as decrease the packet loss ratio.

V. Conclusions

In this work, we present the hardware realization of a scalable queue management. It is relative simpler to implement in hardware than linked-list approach. With the demand towards higher-speed and high bandwidth, the Gbps LAN switching is highly desirable. To design such a high speed system, it needs a more simple and efficient queue management scheme. The proposed implementation architecture has a regular structure that well suits to the VLSI design. The number of virtual connections can grow flexibly since a large cell buffer capacity can be accommodated by cascading the chips. Moreover, by connecting chips in parallel, a large cell pool can be supported. The realiza-

tion approach should be able to accommodate the real-time packet streams.

References

[1] A. Agrawl, A. Raju, S. Varadarajan and M.A. Bayoumi, *A Scalable Shared Buffer ATM Switch Architecture*, VLSI, 1995. Proc. pps 256-261.

[2] Yuhua Chen and Jonathan S. Turner, *Dynamic Queue Assignment in A VC Queue Manager for Gigabit ATM Networks*, IEEE ATM Workshop'98, Proc. pps. 3-10.

[3] Massoud R. Hashemi, Alberto Leon-Garcia, *A General Prupose Cell Sequencer/Scheduler for ATM Switches*, In Proc. IEEE INFOCOM'97, pp. 29-37, March 1997.

[4] Yu-Sheng Lin and C. Bernard Shung, *Queue Management for Shared Buffer and Shared Multi-buffer ATM Switches*, In Proc. IEEE INFOCOM'96, pp. 688-695, March 1996.

[5] T. Kozaki, *etal.*, *32× 32 Shared Buffer Type ATM Switch VLSI's for B-ISDN's*, IEEE Journal on Selected Areas in Communications, vol. 9, pp. 1239-1247, October 1991.

[6] Rudiger H. Hoffmann and Rudi Muller, *A Multifunctional High-Speed Switch Element for ATM applications*, IEEE Journal of Solid State Circuits, vol. 27, pp. 1026-1040, July 1992.