A distributed memory management for high speed switch fabrics

MEYSAM ROODI[+],ALI MOHAMMAD ZAREH BIDOKI[+], NASSER YAZDANI[+], HOSSAIN KALANTARI[+],HADI KHANI[++] and ASGHAR TAJODDIN[+]

+ Router Lab, ECE Department, Faculty of Engineering
University of Tehran, Tehran
++Islamic Azad University of Garmsar, Garmsar
IRAN

Abstract. There are high demands for high capacity switch fabrics due to the data explosion in the internet and exponential growth in the communication link rates. Different architectures have been proposed to design and build switches. Nevertheless, it sounds that the shared memory architecture switch fabrics are optimum in delay and throughput due to the sharing memory among outputs. Furthermore, they support multicast and IP packets efficiently. In this paper, we report the design and implementation of a switch fabric memory manager, which has been implemented, on FPGA, VIRTEX II from Xilinx. The scheduler is completely distributed, thus multicast prioritized packets can be supported easily with the most possible speed. Likewise, the memory manager is scalable and can scales up to two. Also the switch support IP packets internally by using linked lists in memory manager. Moreover, we compare the switch with other switch fabrics, supporting multicast packets in inputs, in delay and throughput. The implementation of scheduler and linked lists are explained in detail and we can reach to 40Gb/s capacity on FPGA. Since outputs work in parallel, implementation of the system is easy and can reach to high speed.

Keywords: shared memory switch, memory management, distributed scheduler, ATM/IP switch, high capacity switching

1 Introduction[1]

Broadband ISDN networks require fast packet switches to transfer IP packets and ATM cells along their appropriate paths. The main purpose of an ATM/IP switch is transferring incoming cells or packets to one or more particular outputs, which is called the output ports related to destinations [1]. Two main methods have been proposed to accomplish the switching function: space division and shared memory [2]. A simple example for space division is crossbar switch, composing of some matrix cross points to create some paths among inputs and outputs. The inputs and outputs in a crossbar switch are connected at cross points in a matrix structure.

The main building block of a shared memory switch [3], [5], [6], [8] is a central dual port memory, which is shared by all input and output ports. Arriving packets on all inputs are multiplexed into a single stream that is fed to the common memory for storage. Inside the shared memory, packets are organized into separate output queues, one for each output port. At the same time, an output stream of packets is formed by retrieving packets from the output queues sequentially. The output stream is then demultiplexed and packets are transmitted on the output ports [2].

Besides the switching function of switch fabrics, queuing is also an important function which needs special attention. Queuing is necessary due to the output contention meaning existing more than one packet destined to an output in a cell time [2]. Three different queuing policies exist in switch fabrics: “input queuing”, “output queuing” and “shared queuing” which buffers packets internally between all outputs.

Input queued switches buffer packets at inputs. Therefore; the memories need only to operate at the line rate. Most IQ switches utilize a crossbar as their switching element. IQ switches, which usually use

[1] This project is supported by Iran Telecommunication Research Center.

FIFOs at their inputs, suffer from a Head of Line blocking (HoL) problem. This problem can be solved b y using Virtual Output Queuing (VOQ), in which there is a separate queue in each input port for each output [13],[9],[4],[11].

Output queuing switches are like the shared memory architecture. Nevertheless, for each output there is a separate memory. Unfortunately, the memory isn't used efficiently in this scheme. Modifying the memory management control circuit makes the shared memory switch flexible enough to perform functions such as priority control and multicast [10].

A nice property of the shared memory switches compared to the input queuing is supporting multicast and IP packets. In input queued switches, the multicast packets are converted to unicast packets and, then; send into switch. In this manner, obviously, the throughput decreases. Furthermore to support IP packets in input switches, they have to assign N^2 queues for inputs and N^2 queues in outputs for assembly and reassembly of cells to packets and vice versa. In shared memory switches, using the port mapping mechanism, supporting multicast and IP packets is simple.

It sounds that the central scheduler of the shared memory is the main bottleneck of the switch fabric. In this paper, we report design and implementation of a new shared memory switch memory management that uses distributed scheduler for managing unicast and multicast packets besides balancing partitioned memory in order to cope with memory bottleneck. Our design is optimized for IPv4 & IPv6 packets with unicast and multicast capabilities. It supports 8 level priority classes with strict priority in outputs. Switching is performed based on segments. Each cell consists of two segments. So the switch can be changed to support different packet formats like frame relay and POS.

Furthermore, our proposed switch scales up to two switches working in parallel. To support this feature, we have to take into account two parameters: the scheduler and the internal cell format. When the switch scales to N, the capacity of switch grows proportional to N. For instance, a switch processing 16 cells in the single mode, must be able to process N*16 cells when it scales up to N. Due to the distributed structure of the scheduler our switch can easily scale to 2 with respect to the base switch. Internal cell format of the switch is based on segments and each cell consists of two segments. Thus; in the input port interface cells are divided into segments and, in the output, they are reassembled into cells.

The rest of the paper is organized as follows: Our internal architecture is presented in section 2. In section 3, we present simulation results. Section 4 concludes the paper.

2 Memory management Architecture

We call the memory management as pointer path occasionally in our design. Pointer path is the heart of the system. It is responsible for receiving headers of input cells, putting each cell in its appropriate output queue and taking cells from output queues with special mechanism and transferring them into outputs. We implement this mechanism using 16 separated linked lists with 800 cells capacity in each output. Separating linked lists enables us to read from and write in each output in parallel. This scheme can handle multicast packets easily and increase the switching capacity of the system. The total shared memory size is **5500** cells. To prevent starvation of an output in a high load (when the shared memory is fully shared, few highly loaded outputs can allocate all of the memory and starve other outputs [13]) and better usage of the shared memory, we choose a middle way. We use at most $\alpha * MEMS$ for each output where MEMS is the size of the shared memory and $1/N \leq \alpha \leq 1$. In our design α is equal to 0.15.

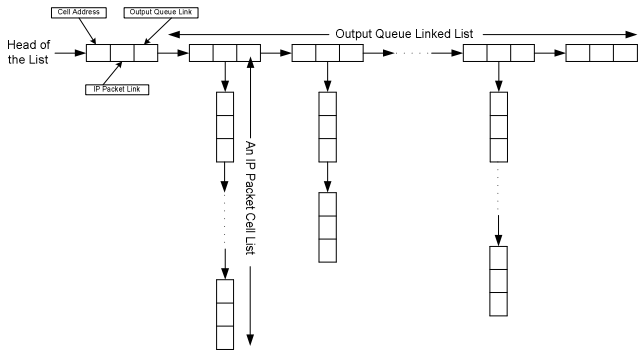

Fig. 1 Two Dimensional Linked List Structure

Our design supports four types of cells. ATM cells and three cell types belonging to the IP packets, which are segmented into the switch internal format. These cell types are starting cell, middle and the last cell of an IP packet. We use a "two dimensional linked list" architecture to handle these types of cells.

ATM cells are put into the appropriate output queues according to their destination addresses and class of service. Since different cells of an IP packet are considered as a whole, they must be sent out sequentially. Therefore, when the first cell of an IP packet departs the switch, others should follow. This conveys that it is not necessary to keep the addresses of the whole IP packet in the output linked lists, but holding the address of the head is sufficient. The internal link of an IP packet is formed using the Fanout/Next memory discussed later in the paper. In this way, we construct a two dimensional linked list as depicted in figure 1.

Figure 2 illustrates the main parts of the pointer path. Write scheduler provides 16 free addresses for the memory (data path) in each cell-time. It also writes the input cell addresses in their appropriate linked lists. Shared Memory Free Address (SMF) contains free addresses of the shared memory. Each cell of Fanout/Next memory stores information of the stored cells in the shared memory which are the number of desired outputs of each cell, in other words its fanout, next cell of an IP packet and cell type. Linked List part includes output queues in which the address of each cell targeting that output is saved. It consists of 16 linked lists, one for each output. In each cell time, read scheduler assigns packets going to outputs. With freeing each cell of memory, its address is added to SMF and Fanout/Next memory is updated. The followings explain these parts individually.

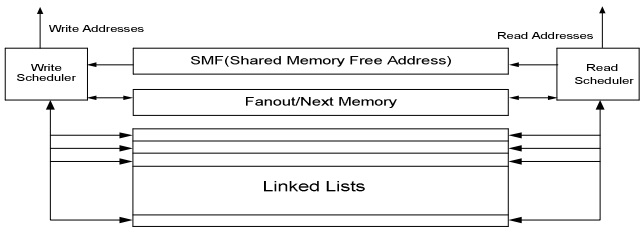

Fig. 2 Pointer Path Block Diagram

Shared memory Free (SMF) Address

This memory contains the shared memory free addresses and is implemented as a FIFO. Its reading port is used by the write scheduler to provide free addresses for the data path (Memory section). Its writing port is used by the read scheduler to return the outgoing cell addresses back to SMF. At the beginning, all of the shared memory addresses are here.

Fanout/Next memory

This memory is equal to the shared memory in the number of memory cells. There is a one-to-one correspondence between each cell of this memory and the shared memory. Each cell of this memory saves the information about a cell that is stored in the same address in the shared memory. The information maintained in this memory is the number of requested outputs, next IP cell address and cell type. Figure 3 shows this memory structure.

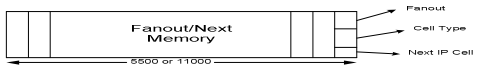

Fig. 3 Fanout/Next Memory

As shown in the figure, each cell of the memory is composed of three fields. The first is the number of requested outputs, the second is the cell type (ATM, start/middle/last cell of an IP packet) and the third is used by IP cells to show the next addresses of the successive cells of an IP packet.

With this architecture, we can support multicast packets in the best fashion. If we use only one linked list for memory management, when all inputs want to send their data to all outputs 16*16 (256) clock cycles are required. Then; we should queue input cells for processing in the coming clock cycles. By using this method, the multicast cells are stored in the shared memory like unicast ones in only one memory location. The address of that memory location is saved in all of its desired output linked lists. Fortunately, as we will see later, this can be done in parallel and in one clock cycle. In addition, the fanout part of the Fanout/Next memory for that address should be set to the sum of its desired outputs. Therefore, the system would know that the cell in that address will be read out "fanout" times.

Write Scheduler

The write scheduler has three minor parts as can be seen in figure 4. These parts are:

- Free Address Provider
- Cell Header Shift Register
- Controller

"Free Address Provider", is responsible for receiving free shared memory addresses from SMF serially and providing them to the data path in parallel. As discussed in [14], the data path requires 16 shared memory free addresses at the beginning of each cell

time. There is a possibility that an input does not have any valid cell in a cell time. Free Address Provider does not write back the unused free addresses in this situation. In other words, once extracted a free address it is held until is used by the data path. Because the cell information should be written when the cell enters the switch, the write address of the Fanout/Next memory is driven by this part too. Additionally, when a sequence of IP packet cells enter the switch, this part link them using Fanout/Next memory.

We have a cell time which consists of at least 16 clock pulses [14], and we can process every input cell entering from one of 16 input ports, in a clock cycle from those 16. Cell Header Shift Register stores 16 cell headers of the input cells and shifts out them for processing. The majority of write data into the Fanout/Next memory (fanout and cell type) are driven from this part.

The last component of the write scheduler is its controller. It drives the write command to the Fanout/Next memory and the linked lists when a valid cell enters the switch. There are 16 different write commands for the linked lists. The controller drives several linked lists write commands in the case of a multicast input cell. Also, it controls the cell type before driving the write commands to the linked lists. Only ATM cells and IP start cell type addresses should be stored in linked lists.

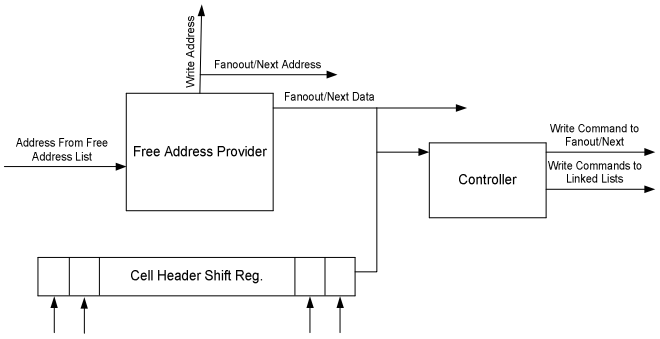

Fig. 4 Write Scheduler Block Diagram

Linked List

The main part of the pointer path is linked lists. F igure 5 shows a diagram of a linked list. 16 blocks of this architecture make up the linked lists part in figure 2, each of them holds the output queue for an output port. The linked list is designed in such a way that it is able to hold and implement 8 separate linked lists, each belongs to a specific class of cells requesting that output, concurrently.

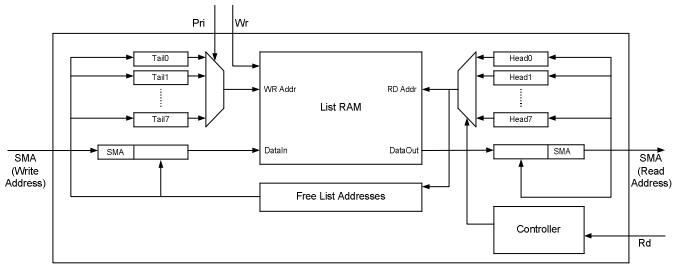

Fig. 5 Linked List Structure

The list RAM, which can be seen in the center of the picture, holds 8 linked lists. As mentioned before its capacity is 800 memory cells. The left side of the picture is mainly responsible for adding a shared memory address to the output queue. When the write scheduler controller drives the write command to a linked list (the wr signal in the above picture), the input SMA (Shared Memory Address) is written at the end of its related class queue. The input class signal drives the select of the List RAM write address multiplexer. There are eight tail registers in which the tail addresses of the 8 linked lists exist. The multiplexer is used to select the suitable tail register, according to the input class. After a write, the related tail register will be updated too. It is replaced by a free List RAM address which is extracted from the Free List Addresses. Free List Addresses is just like the SMF block in the main architecture and its duty is the same. It should be noticed that shared memory addresses are treated as the data parts of the linked lists.

The right side of the linked list block diagram is in charge of reading a shared memory address which its contents should be sent out from the related output port. It is very similar to the write part; the 8 head registers store the head of the linked lists and a multiplexer selects among them. One important note about this part is that a controller makes decisions about the outgoing addresses. Although the decision policy which is used in this architecture is "strict priority", every other policy can be used without any major changes. This is because the controller can be changed to reflect the new course of actions without affecting the other parts. The current decision strategy is that the controller checks all of the class linked lists from the highest priority. The first linked list which is not empty is chosen ignoring the others.

The last point worth noting is that the read part of the linked list does its duty after receiving a read command from the read scheduler. This is necessary because there are circumstances that any address should not to be read. We will see this in the next section.

Read Scheduler

The remaining part of the pointer path is the read scheduler. The decision making among different classes is done inside the linked lists as explained before. Read scheduler has some elements itself and some other functionality.

Read scheduler consists of three components. The first part is 16 "port read schedulers" and a multiplexer to select among them. Every of these "port read schedulers" belongs to one of the linked lists. Every "port read scheduler" receives the shared memory address from its related linked list in every cell time and stores it. Then, it receives and saves the information of the shared memory address obtained from the Fanout/Next memory. The multiplexer directs the stored addresses of the different port read schedulers to the Fanout/Next memory address port, each one in a clock cycle. It processes the cell type of the memory address. In the case that a cell type is the start cell of a long IP packet the next IP address, which is a field of the Fanout/Next memory content, is stored and used in the coming cell time. The read command to the linked list driven by the "port read scheduler" is not activated after a start cell of an IP packet. In other words, the task of sending out cells is done by the "port read scheduler" when an IP packet is going to leave the switch. Consequently, when the flow of IP packet cells finishes, in other words the current cell type is the last cell of an IP packet, "port read scheduler" reactivates the read command to the related linked list to continue the read procedure from the output queue.

Figure 6 illustrates a very simple block diagram of a typical "port read scheduler". With respect to the picture, every block has a register to hold the next IP cell address. The controller controls the select signal of a multiplexer to either feed the "Read Address" from the coming address from the linked list or the stored next address register. In the latter case, the controller does not activate the read signal to its linked list (Rd). Furthermore, it has an input from the control path part (BPIn), which simply deactivates that output port.

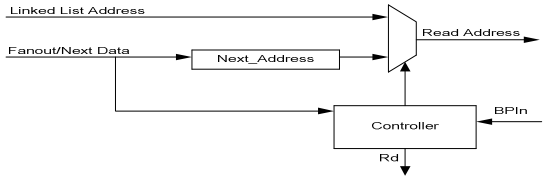

Fig. 6 Port Read Scheduler

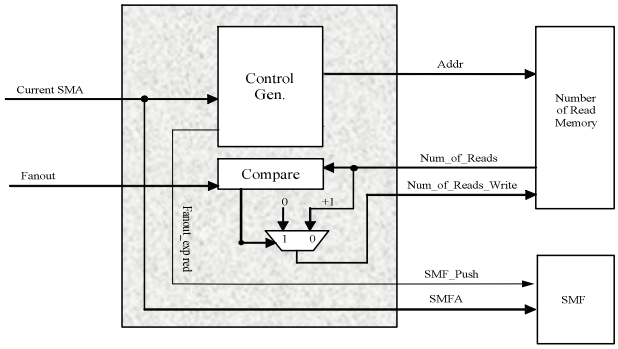

Fig. 7 Return Shared Memory Address

Since the system supports multicast cells, we should use a block named "Return Shared Memory Address" in picture 7. It has a memory just like the Fanout/Next memory which has identical memory cells to the shared memory and there is a one to one correspondence between the memory cells of this memory and the shared memory ones. In the cells of this memory, the number of times that a memory cell has been read is stored. Whenever a cell is read out from the shared memory and sent out, the number of reads from that cell incremented by one and the result is compared to the fanout of that cell. If they are equal, the cell is read "fanout" times and its address can be returned back to SMF. Otherwise, we should not return back the cell address, because its contents will be used in future. In figure 7, we give a picture of the "Return Shared Memory Address". As it can be seen from the picture, when the Shared Memory Address is ready to be returned to SMF, its associated number of reads is reset to zero. This is because a new cell is coming to be written in that address.

3 Simulation

Few simulations have been done using a software model of the architecture, in order to study different system parameters such as throughput and delay. The simulation system works based on the discrete event. All programs have been written in C++ and implemented with the Visual C++ compiler.

First, we compare the proposed architecture (distributed scheduler) to switches that convert a multicast packet to some unicast packets in inputs. F igure 8 shows the delay and throughput of our switch compared to the central switches that support multicast in inputs like [7, 12] for multicast packets. However, when packets are unicast both schedulers behave the same.

The picture illustrates the fact that our distributed scheduler supporting multicast packets internally performs better in throughput and delay especially for bursts and IP packets. The reason is the distributed scheduler that manages multicast packets on the fly, like unicast packets. For example, in a multicast packet with 1 and 3 destination outputs, in the distributed scheduler the address of packets will be put in output queues 1 and 3. But in central scheduler, first, the packet will be sent to the output queue 1 and, then, to the output queue 3. We can conclude that the internally allocated memory for one packet in a distributed scheduler is less than the other one, thus, the memory is used optimally.

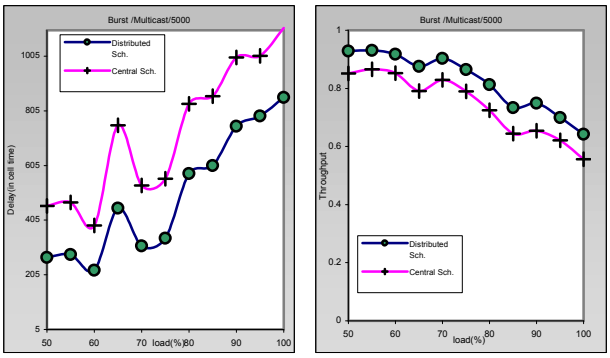

Fig.8 The delay and throughput of distributed and central switch with multicast Bursty traffic.

4 Conclusion and Future works

In this paper, we present a 40 Gbps ATM/IP switch fabric memory manager. The advantage point of this switch memory manager is its scheduler. The scheduler is distributed, which means we have 16 independent linked lists for managing output queues. Thus, supporting multicast packets in high speed is simpler while the switch clock and capacity can be increased. Furthermore, the switch scales up to two parallel switches. In addition, the scheduler supports IP packets naturally using separated linked lists and has special queue for each class of priority. The simulation results show that in comparison to the other switches its delay and throughput for multicast packets are better. Furthermore, the implementation of the switch is simple and can reach to high capacity very easily. The internal switch format is segment base implying that each cell is composed of two segments. Then the switch can support other network technologies like frame relay and POS easily.

References

1. M. Schwarz, Broadband Integrated Networks, Prentice Hall, Inc. 1996.
2. F. A. Tobagi, “Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks,” Proc. IEEE, vol. 78, no. 1, Jan. 1990, pp. 133–67.
3. A.M. Zareh, N. Yazdani, V. Azhari and S. Samadian, " A High Speed ATM/IP Switch Fabric Using Distributed Scheduler," Springer-Verlag LNCS 2662, 2003.
4. Anderson, T.; Owicki, S.; Saxe, J.; and Thacker, C.; “High speed switch scheduling for local area networks,” ACMTrans. on Computer Systems. pp. 319-352. November 1993.
5. Andersson, P., and C. Svensson, “A VLSI Architecture for an 80 Gb/s ATM Switch Core,” in *Proc. 8th Annual IEEE Int’l Conf. Innovative System in Silico*n, Austin, TX, Oct. 9-11, 1996.
6. Denzel, W.E., A.P.J. Engbersen and I. Iliadis, “A Flexible Shared-Buffer Switch for ATM at Gb/s Rates,” *Computer Networks and ISDN System*s, vol. 27, no. 4, Jan. 1995, pp. 611-624.
7. Iliadis, I. and W.E. Denzel, “Analysis of Packet Switches with Input and Output Queuing,” IEEE Trans. Commun., vol. 41, no. 5, May 1993, pp. 731-740.
8. Katevenis, M., D. Serpanos and P. Vatsolaki, “ATLAS I: A General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control,” in *Proc. IEEEHot Interconnects IV Symposiu*m, Stanford, CA, Aug. 15-17, 1996.
9. A.M. Zareh, V. Azhari and N. Yazdani, " A Decomposed Hierarchical Logarithmic Scheduling Algorithm for Input-Queued Switches ," in Proceedings of IEEE ICT’03, Feb. 2003.
10. N. Endo et al., “Shared Buffer Memory Switch for an ATM Exchange,” IEEE Trans. Commun., vol. 41, no. 1, Jan. 1993, pp. 237–45.
11. Mckeown, N.W., “Scheduling Algorithms for Input-Queued Switches,” Ph.D. Thesis, University of California at Berkeley, 1995.
12. Minkenberg, C.J.A., “On Packet Switch Design” Ph.D. Thesis, 2001.
13. Tam, Y. and G.L. Frazier, “Dynamically Allocated Multi-Queue Buffers for VLSI Communication Switches,” *IEEE Trans. Computer*s, vol. 41, no. 6, Jun. 1992, pp. 725-737.
14. N. Yazdani, A.M. Zareh, M. Roodi, H. Kalantari, A. Tajoddin and H. Khani"Design and implementation of a 40 Gb/s ATM/IP switch on FPGA" , submitted to 12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB & PGTS 04), Dresden Germany, Sep. 12-15, 2004.