Design and Implementation of High-Speed Arbiter for Large Scale VOQ Crossbar Switches

Chun Kit Hung, Mounir Hamdi, and Chi-Ying Tsui
Hong Kong University of Science and Technology
Clear Water Bay,
Kowloon, Hong Kong

ABSTRACT

Crossbars are frequently used as the switching fabric for high-performance packet switches (IP routers, ATM switches, Ethernet switches). The performance, functionality, and scalability (in terms of line rate and/or number of ports) of these switches are directly related to the arbitration/scheduling algorithm which must retrieve the state information of input queues, compute a (pseudo-) optimum matching, and configure the crossbar accordingly, all within one packet cycle. In this paper, we give a detailed hardware design and implementation of a novel arbitration scheme, named RDSRR [1], that lends itself well to high-speed implementation, while at the same time achieves excellent performance under a variety of traffic patterns. We present a novel pipeline technique and the full-custom design of the arbiter circuit using TSMC 0.25 m CMOS technology which can support switch sizes of up to 256 x 256 at a line rate of 10 Gbps.

1. INTRODUCTION

Most of the academic research and the commercially available crossbar switches are based on virtual output queuing (VOQ). Using VOQ, at each input a separate FIFO queue is maintained for each output. After a forwarding decision has been made, arriving packets are placed in the queues corresponding to their outgoing ports. An arbitration algorithm then examines the contents of all the input queues, and finds a conflict-free match between inputs and outputs. As a result, these arbitration/scheduling algorithms are key to the performance, scalability, and hardware complexity of VOQ switches.

Numerous scheduling algorithms, such as PIM, iSLIP, RPA, FIRM, and DRRM, that use parallelism, arbiters, and iteration have been proposed for VOQ crossbar switches [2], [3], [4] [5], [6]. Briefly, these arbitration algorithms iterate the following steps until no more requests can be accepted (or for a given number of iterations):

1. *Request:* Each unmatched input sends a request to every output for which it has a queued cell.

2. *Grant (outputs):* If an unmatched output receives any request, it grants one by selecting (in some fashion) a request uniformly over all requests.

3. *Accept (inputs):* If an unmatched input receives grants, it accepts one by selecting (in some fashion) an output randomly among those grant to this input.

The functional differences among the various algorithms exist only in the way the outputs choose which input to issue the grant to, and in the way the inputs choose which grant to accept. However, it can have a huge difference in the performance of the switch [6].

Through careful examination of the various algorithms, it has been found that the performance and complexity of these arbitration algorithms can be attributed to two key factors: desynchronization of the output arbiters and efficient pipelining schemes [3]. The desynchronization of output arbiters is essentially making the outputs point to different inputs. We know that if several outputs grant the same input, no matter how this input chooses, only one match can be made, and the other outputs will be idle. To get as many matches as possible, it is better that each output grants a different input. Since each output will select the highest priority input if a request is received from this input, it is better to keep the output pointers desynchronized. As for pipelining, by making the request, grant, and accept stages busy all the time, that would allow us to iterate the algorithm for more iterations. This in turn improves the performance of the matching process.

In this connection, we used an arbitration algorithm, RDSRR, that achieves the best desynchronization effect [1]. RDSRR was shown to outperform state-of-the-art related scheduling algorithms, and by using intelligent pointer-movement schemes it performs well under any traffic pattern. RDSRR scheduling algorithm can also simplify the arbiter design as the pointers update mechanism in both input and output sides are always increase by one. As a result, the pointer circuitry can be easily embedded into the arbiter design which can greatly reduce the wiring area in the layout. The other feature of RDSRR is that the searching direction of the output arbiter will be changed from clock-wise to counter-clockwise direction alternatively at the beginning of each cell time. This feature can have a better desynchronization and further enhance the performance of the RDSRR scheduling algorithm.

In this paper, we present the design and implementation of the arbitration scheme for the RDSRR algorithm. First we propose a novel pipelining scheme for the design of the RDSRR. In essence, we are able to achieve one more iteration of the algorithm using the same amount of hardware/time complexity. Secondly we also present the efficient circuit design techniques that minimize the delay bottleneck of the hardware design and at the same time can support large number of ports.

2. RDSRR Scheduler Architecture

Figure 1 shows the overall architecture of the RDSRR scheduler which is similar to that of iSlip[7]. The three blocks represent the request phase, the grant phase, and the accept phase of the algorithm. The request blocks are used to store and forward the incoming request vectors to the grant blocks. After the request vectors pass through the grant stage and the accept stage, the

0-7803-7761-3/03/$17.00 ©2003 IEEE

scheduler will make a decision by selecting which input and output ports should be connected. The successive iterations of the algorithm can help increase the number of input/output matching during each cell time. A feedback loop is used to mask off those requests that have been accepted in previous iterations. The scheduler will not consider the accepted requests again during the next iteration. In fact, the request blocks are registers, and we can simply treat them and the grant blocks as a single unit.

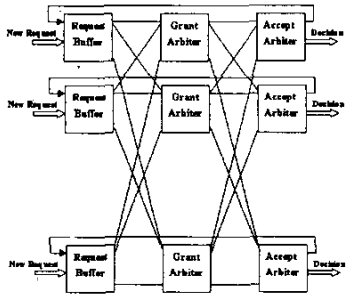

Figure 1. Overall architecture of the scheduler.

3. Proposed Pipelining Scheme

A proper pipelining scheme can increase the throughput of the whole scheduler. Figure 2a shows the pipelining scheme which is employed by the iSLIP arbiter that has been implemented in the Tiny Tera experimental switch designed by Stanford University [7].

This figure shows that three iterations can be executed within each cell time. After the third iteration, a setup up time is needed to update the pointers and the request vectors for the next cell time slot.

In this paper we propose a more optimal pipelining scheme. This is shown in Figure 2b.

As shown in Figure 2a, after the third iteration of the first cell time the grant arbiter will be in an idle state. It will be used to schedule a new connection until the beginning of the second cell time. Similarly, the accept arbiter will be in an idle state at the beginning of second cell time. It will start to make the connection decision after the grant arbiter made a decision in the first iteration of the second cell time. As a result, these idle states will decrease the efficiency of the whole scheduler.

By carefully examining our proposed pipelining scheme, as shown Figure 2b, and the pipelining scheme for the Tiny Tera, we can see that our proposed scheme can execute one additional iteration of the algorithm within the same amount of time. This is significantly important, since increasing the number of iterations of the arbitration algorithm can greatly improve the performance of the switch.

In this proposed scheme, the request vectors will be updated after the third and the fourth iterations during each cell time. As a result, the grant arbiter (in the second cell time) can take in the updated data after the third iteration of the first cell time.

The new data taken in during the first iteration of the second cell time have ignored the acceptance status in the fourth iteration of the first cell time. As a result, we have two cases that we need to address.

1. There are no more acceptances in the fourth iteration of the first cell time. If this case happens, it means that the updated data taken by the grant arbiter during the second cell time are correct. Hence, no more acceptances in the forth iteration implies that the crossbar configuration is the same after the third and the fourth iteration.

2. There are some acceptances in the fourth iteration of the first cell time. If this happens, it means that the updated data taken by the grant arbiter during the second cell time may be incorrect. This is due to the fact that the request vectors taken by the grant arbiter have not yet been updated by the accept arbiter during the fourth iteration of the first cell time. In this case empty cell may be sent across the switch fabric.

However, the probability of sending empty cell is low. It is because if there are any acceptances in the fourth iteration of the first cell time, it means that the VOQ of these "just accepted" signal(s) cannot be updated for the next cell time and the request vector value for these VOQs will still be "1". Only when these "just accepted" VOQs do not have any cell waiting for scheduling in the next cell time then empty cell will be sending across the crossbar switch. The impact of this on the overall scheduling performance will be insignificant when the switch is under heavy traffic as the VOQs are seldom empty.

Simulation results shows that our proposed scheme gives a higher performance than the scheme used in Tiny Tera. The simulation results will be shown in section 6.

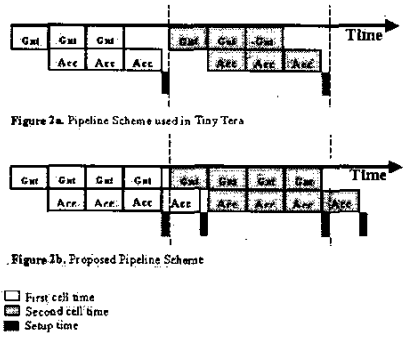

Figure 2. Pipelining schemes used in iterative scheduling algorithms.

4. Arbiter Hardware Design

In this section, we will illustrate the hardware design of the RDSRR arbitration algorithm. Briefly, an arbiter schedules the crossbar connections based on the incoming request vectors and the pointer update mechanism. In particular, the most critical hardware design issue of the arbiter is the design of the priority encoder [7]. As a result, designing a high-speed priority encoder can reduce the arbitration time significantly. In the following we will introduce the design of a high-speed arbiter architecture.

Both of the iSLIP and RDSRR arbitration algorithms base their scheduling decision on the round-robin searching mechanism. These two scheduling algorithms will start the search based on the current pointer position. Whenever it reaches the last element, it

will jump back to the first element and search again. Obviously, this round-robin searching mechanism will form a loop. Therefore, we can apply a similar technique to break this feedback loop just like what has been done for the iSLIP design in the Tiny Tera. Here two priority encoders are used. The first one is responsible for the search start at the beginning element until the end element. While the other one is responsible for the search starting at the current pointer position until the end element (figure 3). As mentioned before, for RDSRR, the pointer is updated by simply increment by 1. Therefore this simple pointer updating mechanism can be embedded within the each arbiter which can reduce the layout area of a scalable arbiter. Another feature for RDSRR is that the pointer search alternates between clock-wise and anti-clockwise direction. To support different directions for the priority encoding, we add a flipping block to mirror-flip the filtered request vector before sending to the priority encoder if anti-clockwise search is needed. The corresponding output is flipped at the output to get the correct grant signal vector.

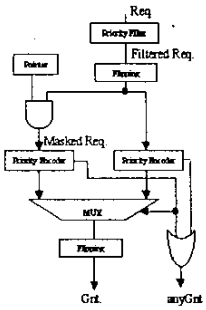

Figure 3. Proposed arbiter architecture.

5. Priority Encoder

In this work we design the scheduler such that it can support large number of ports, such as 256X256. In order to support a 256x256 switch, the priority encoder should be fast enough to trace through the 256 bits request vector. Here we use a hierarchical tree structure and domino NOR gates to improve the speed of the 256 bit priority encoder (PE). The overall architecture of the priority encoder is shown in Figure 4 which shows the architecture of a priority encoder with 16 inputs. Priority encoder with 256 inputs can be done in a similar fashion. The 16-bit request vector input to the PE is separated into four groups (4 bits in each group) where each group is processed by the "4 bits Priority Encoder". Figure 5 shows the detailed structure of this 4-bit priority encoder. For example, if "0011" is fed into a, b, c and d respectively. It will return "0010" for the 4 bits grant signal and it will give a "1" on the anyGnt signal.

This circuit can trace the first "1" appearing in the request vector in parallel, the output will show the bit position of this "1", that is "0010". The anyGnt signal will return "1" if there is any "1" appearing in the request vector.

After four groups of 4 bits vectors pass through the 4-bit priority encoder, we consider all the processed results by using the anyGnt signals.

If all the 4 bits vectors contain at least one "1" in the request vectors, all the four anyGnt signals are "1" (that is, 1111). These anyGnt signals will be fed into the controller. The controller, as shown in Figure 6, will use these signals to test if the previous 4-bit priority encoder has any grant signals or not. If it has, it will set the remaining grant block signals to zero vectors. The OR gate in the 4-bit priority encoder, the NOR gate in the controller, and the multiplexer form the critical path of the hardware design.

This structure can be easily extended to a 256-bit priority encoder by using sixteen 16-bit priority encoders (each based on the structure shown in Figure 6) and 16 bits controller. In this case, the maximum fan-in for each OR gate and NOR gate will increase. However by using a 16-bit dynamic NOR gate structure, the loading is reduced and the speed is improved.

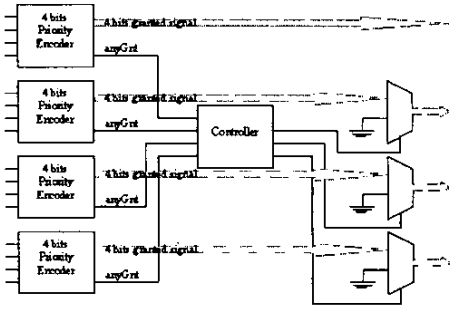

Figure 4. A 16-bit priority encoder architecture.

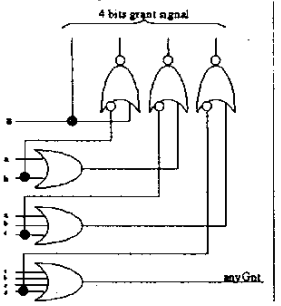

Figure 5. The design of 4-bit priority encoder.

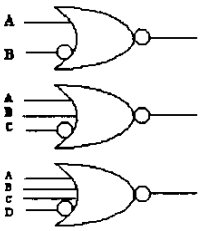

Figure 6. The design of the controller used in 16-bit priority encoder.

6. Experimental Results

The proposed pipeline structure can be used in any maximal matching scheduling algorithms, e.g., iSlip and RDSRR. To verify the effect of the proposed pipeline structure, we simulated the RDSRR scheduling algorithm using the pipeline structure and compared the performance with that using the original pipeline structure employed in Tiny Tera. Comparisons were done for three different traffic models: uniform Bernoulli i.i.d., uniform bursty and unbalance traffic. The switch evaluated is a 128 x 128 switch.

The results were collected between 50,000th cell time slots and 500,000th cell time slots. The number of iteration is assumed to be one. The results are summarized in Figures 7-9.

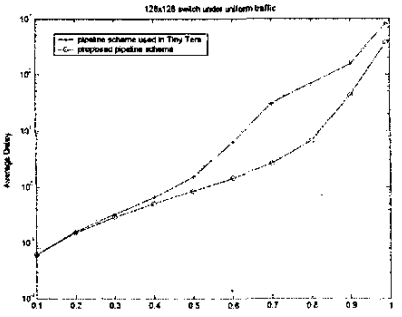

Figure 7. Average delay under uniform traffic

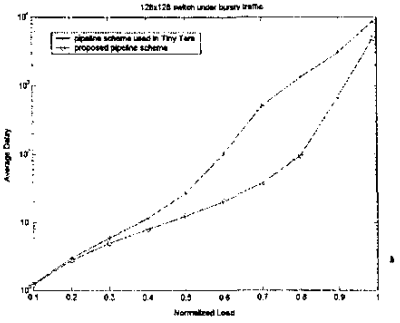

Figure 8. Average delay under bursty traffic

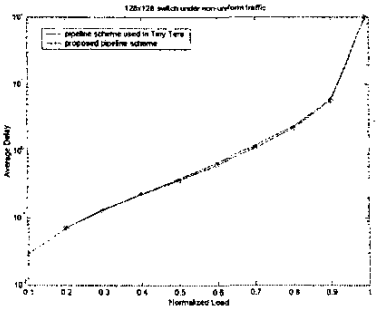

Figure 9. Average delay under non-uniform traffic

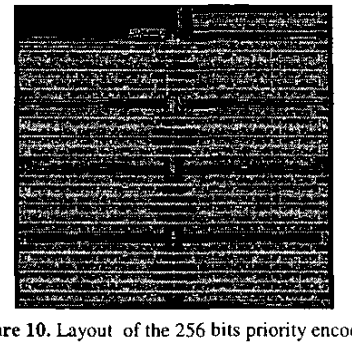

Figure 10. Layout of the 256 bits priority encoder.

Figures 7 and 8 show that the proposed pipeline scheme has a significant improvement under Bernoulli i.i.d. uniform and uniform bursty traffic while Figure 9 shows the proposed scheme gives a little improvement under the non-uniform traffic.

Figure 10 shows the layout of the 256 bits priority encoder using TSMC 0.25 m SCN5M_DEEP (lambda=0.12) CMOS technology. The size of the layout and the post layout HSPICE simulation result are shown in table 1. Results are simulated under typical process corner, normal condition with 2.5V power supply voltage.

	256 bit Priority Encoder
Layout Size	292 m x 273 m
Post Layout Simulation delay	2.7 ns

Table 1. Layout size and post layout simulation of the 256 bits priority encoder.

7. SUMMARY

In this paper, we have proposed a novel pipelining scheme for our previously designed RDSSR arbitration algorithm. This pipelining scheme allows us to iterate the algorithm for one more iteration when comparing with conventional pipelining schemes. As a result, significant improvement in performance can be achieved. In addition, we have presented the detail hardware design for this arbitration scheme. From the experimental results it is shown that we can easily support lines rates of OC 192 and large switch size of up to 256 x 256.

8. EFERENCES

[1] Y. Jiang and M. Hamdi, "A Fully Desynchronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture," *International Workshop of High-Performance Switching and Routing*, Dallas, 2001.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High Speed Switch Scheduling for Local Area Networks," ACM Trans. Comput. Syst., pp.319-52, Nov.1993.

[3] N. Mckeown, "Scheduling Cells in an Input-Queued Switch," PhD thesis, Uni. of California at Berkeley, May 1995.

[4] D.N. Serpanos, and P. I. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switch," *Proc. IEEE ATM Workshop*, May 1998.

[5] N. McKeown , "iSLIP: A Scheduling Algorithm for Input-Queued Switches," *IEEE Transactions on Networking*, Vol 7, No.2, April 1999.

[6] M. G. Ajmone Marsan, A. Bianco, E. Fllippi, and E. Leonardi, "On the Behavior of Input Queueing Architectures," *Eur. Trans. Telecommunications*, Vol. 10, No. 2, pp. 111-124, Mar. 1999.

[7] P. Gupta and N. McKeown, "Design and Implementation of a Fast Crossbar Scheduler," *IEEE Micro Magazine*, Jan 1999.

[8] Y. Jiang. *Packet Scheduling Algorithms for Virtual Output Queued Switches*. MPhil thesis, Department of Computer Science, Hong Kong University of Science and technology, Hong Kong, 2001.