Design and Implementation of Scheduling schemes for Input-queued Packet Switches, as a Senior Design Project Experience

Roberto Rojas-Cessa, Robert Nawrocki, Darine Abisaleh, Ankita Raut, Pratik Sheth, Suhasini Madidi, Justus Srigiri, Shukri Abdelhalim, and Mohammed Abumokhemar,
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102
Email: rrojas@njit.edu

***Abstract*— The use of virtual output queues (VOQs) in input-queued (IQ) switches can eliminate the head-of-line (HOL) blocking phenomena that limits matching and switching performance. An effective matching scheme for IQ switches with VOQs must provide high throughput under several admissible traffic patterns while keeping the implementation feasible. A variety of matching schemes that deliver high throughput under traffic with uniform distributions has been proposed. In this document, we present the design and implementation of diverse matching schemes for input-queued matching schemes. These implementation have been performed as undergraduate senior projects.**

***Index Terms*— input-queued switch, round-robin matching, virtual output queue, captured frame, eligible frame**

I. Introduction

Input-queued (IQ) switches have been objective of research for several years as they work without the speedup requirement of an output-queued (OQ) switch. Because of the feasible implementability with current technologies, IQ switch architectures have been adopted by several manufacturers for switch/routers. The introduction of virtual output queues (VOQs), where one queue per output port is allocated at an input port of IQ switches, is known to remove the head-of-line (HOL) blocking problem from IQ switches. HOL blocking causes idle outputs to remain so even in the existence of traffic for them at an idle input [1].

Roberto Rojas-Cessa is the corresponding author. He is a Faculty of the Department of Computer and Electrical Engineering, New Jersey Institute of Technology, University Heights, Newark NJ 07102 USA. Email: rrojas@njit.edu.

The well-known Head-Of-Line (HOL) blocking problem [2] is present on this architecture, which uses First-In Fist-Out (FIFO) buffers as input queues. HOL blocking makes 100% throughput unachievable [2], consequently, QoS guarantees are undefined. The disadvantage to using a single FIFO as an input buffer is that only an HOL cell per input is considered by the scheduler; the matching choices are largely reduced, producing blocking of cells that could be directly routed. Some solutions have been proposed to solve HOL blocking.

A way to solve the HOL blocking in IQ switches is by providing a single and separate FIFO for each of the outputs that an incoming cell could be destined to. This is called a Virtual Output Queue (VOQ). A scheduler might have different matching options to accomplish the most profitable or largest match. Figure 1 shows a packet switch with VOQs at the input ports and a scheduler controlling the forwarding of cells from the VOQs to the output ports.

It is common to find the following practices in packet switch design: 1) Segmentation of incoming variable-size packets at the ingress side of a switch to perform internal switching with fixed-size packets, or cells, and re-assembling the packets at the egress side before they depart from the switch. 2) Use of VOQs, to avoid head-of-line (HOL) blocking [1]. 3) Use of crossbar fabrics for implementation of packet switches because of their non-blocking capability, simplicity, and market availability. This paper follows these practices.

One major requirement for an input-queued switch is the delivery of high throughput under different traffic conditions. We consider admissible traffic [4], with Bernoulli arrivals and uniform and nonuniform distribu-

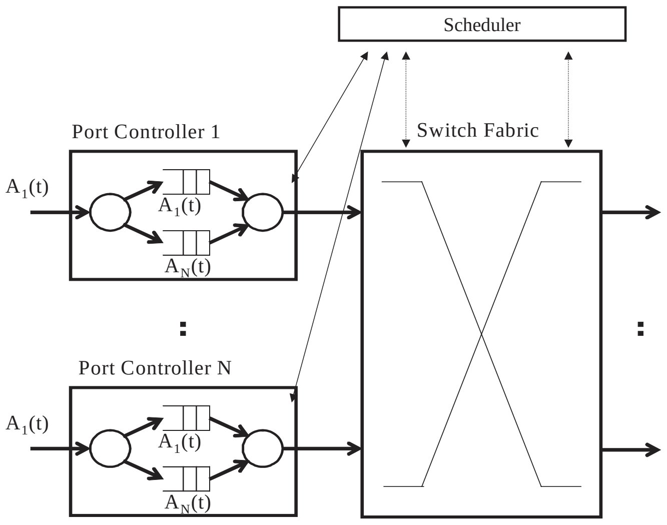

Fig. 1. Switch with VOQs and scheduler

tions. An IQ switch, based on a crossbar switch fabric and VOQs, has the throughput performance dependable mainly on the used matching scheme and speedup. In general, matching schemes are required to provide: a) low complexity, b) fast contention resolution, c) fairness, and, d) high matching efficiency.

Maximum weight matching (MWM) have been used to show that IQ switches with VOQs can provide 100% throughput under admissible traffic [4]. These matching schemes show that using a quantitative differentiation among contending VOQs, based on queue occupancy or cell waiting time, can make a switch deliver 100% throughput. However, MWM schemes have intrinsically high computation complexity that is translated into long resolution time and high hardware complexity. This makes these schemes prohibitely expensive for a practical implementation with current available technologies. An alternative is to use maximal-weight matching schemes, also based on quantitative differentiation of queues. However, the hardware and time complexity of these schemes can be considered high for the ever increasing data rates, and a large number of iterations may be needed to achieve a satisfactory matching result. Moreover, weight-based schemes may starve queues with little traffic to provide more service to the congested ones, therefore, presenting unfairness [7].

Maximal-size matching are another way to resolve contention in IQ switches [5]. Schemes based in round-robin matching, such as iSLIP [9], DRRM [10], and SRR [12] have been proposed to deliver 100% throughput under uniform traffic. iSLIP showed that the desynchronization effect, where arbiters reach the point where each of them prefers to match with different input/outputs, is beneficial for switching under uniform traffic. Other schemes have employed further the advantage of this effect [12], [13].

In this paper, we describe several matching schemes for input-queued packet switches. Two of the presented schemes are round-robin based, and the other is time-stamp based. The former two schemes belong to a maximal-size matching class, while the latter belongs to a maximal-weight matching class. The implementation of these schemes have been undertaken as undergraduate senior design projects. Our contribution is to share the experience obtained by undergraduate students with digital design using a very-high description language, such as VHDL.

This paper is organized as follows. Section II presents the switch model under study and definitions used in this paper. Section III introduces the iterative round-robin matching scheme. Section IV introduces the iSLIP matching scheme. Section V presents the iterative oldest-cell first matching scheme. Section VI presents the results of the development of this project. Section VII presents the conclusions.

II. Switch Model and Preliminary Definitions

We consider an IQ switch with N input output ports. There are N VOQs at each input port. A VOQ at input port i that stores cells for output port j is denoted as $VOQ(i, j)$.

any of the scheduling algorithms under study were designed for **input-buffered crossbar** architectures of size N x N, where N is the number of input and output ports. This architecture has the following properties:

1) Every input port has some buffer structure to hold arriving traffic until it can be transferred to its output.
2) Every input port has the data rate, defined as R.
3) During a matching phase, a set of cells is chosen to satisfy the so-called **crossbar constraint**: at most one cell can leave an input port and at most one cell can enter an output port as a result of a matching.

Considering that cells can be momentarily stored at some buffers (e.g., input buffers) to avoid input or output contention, a set of cells that can be transferred has to be chosen. Thus, the selection of this set of cells has to be made by an arbiter, which has to find a match between the input and output ports. The way to describe the type of scheduler is the matching algorithm used.

We can classify scheduling algorithms as being maximum or maximal. A **maximum match** is one that matches the maximum number of inputs and outputs. This kind of match algorithm can be split into two categories, size or weight matching.

Maximum size match. As in [3], let $S_{i,j}$ be a service indicator such that $\sum_{i=1}^{N} S_{i,j}(n) \leq 1$ and $\sum_{j=1}^{N} S_{i,j}(n) \leq 1$; a value of 1 indicates that input i is matched to output j or that queue $Q_{i,j}$ is allowed to forward a cell to its output. It is the match that maximizes the number of connections $\sum_{i,j} S_{i,j}(n)$.

Maximum weight match

This maximizes the total weight,

$$\sum_{i,j} S_{i,j}(n) w_{i,j}(n).$$

In this kind of matching, a weight is assigned to every input-output pair according to a considering parameter that assigns priorities. A weighted graph is shown in Figure 2.

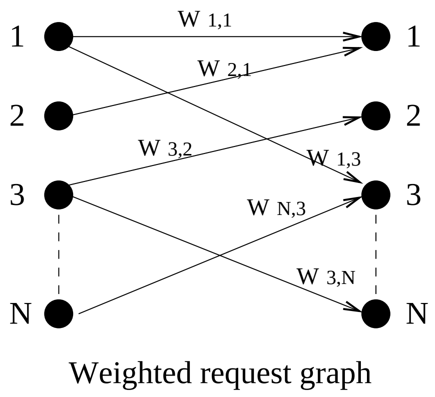

Fig. 2. Request from VOQ at the inputs can be seen as a weighted request graph

A **maximal match**, as in a bipartite graph, is the match where no more matches can be made trivially. Using the crossbar constraint, a **stable matching** is then defined: A matching of an input port to an output port is said to be stable if for each cell c waiting in an input queue, one of the following cases holds:

1) Cell c is part of the matching, i.e., c will be transferred from the input side to the output side during this phase.
2) A cell that is ahead of c in its priority list is part of the matching or a cell that is ahead of c in its output priority list is part of the matching.

An example of this definition is shown in Figure 3. These three cases are depicted in a 3 x 3 switch, where the cells are identified by the input port number and the output port letter. The cells 1B and 3A are included in the match. This is case 1 of the stable match definition. However, for case 2, cell 1C could not go because the cell 1B from its input was matched. Similarly, for case 3, cell 2A could not go because its output received cell 3A, which is from another input. Furthermore, cell 3B could not take part in the match because another cell from its input, 3A, was matched instead. Also, another cell from another input, cell 1B, was matched to 3A's destination, port B. This demonstrates that cases 2 and 3 can apply at the same time.

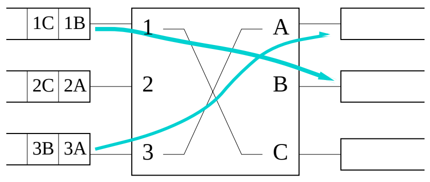

Fig. 3. Example of the stable matching case

III. Iterative Round-Robin Matching (iRRM)

This scheme, presented in [8], is the simplest version of any matching scheme based on round-robin selection. iRRM offers an acceptable performance in just two iterations. For the simplicity of its algorithm, iRRM offers a very simple implementation. There exists a scheduler in each input and output port that is able to select a pairing port. These schedulers or pointers are named accept pointer a_i at the inputs and grant pointer g_i at the output. The algorithm is as follows:

1) Each unmatched input sends a request to every output for which it has a queued cell.
2) If an unmatched output receives any requests, it chooses the one that appears next in a round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer g_i to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the granted input.
3) If an input receives a multiple grant, it accepts the one that appears next in a round-robin schedule starting from the highest priority element. The pointer a_j to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the accepted output.

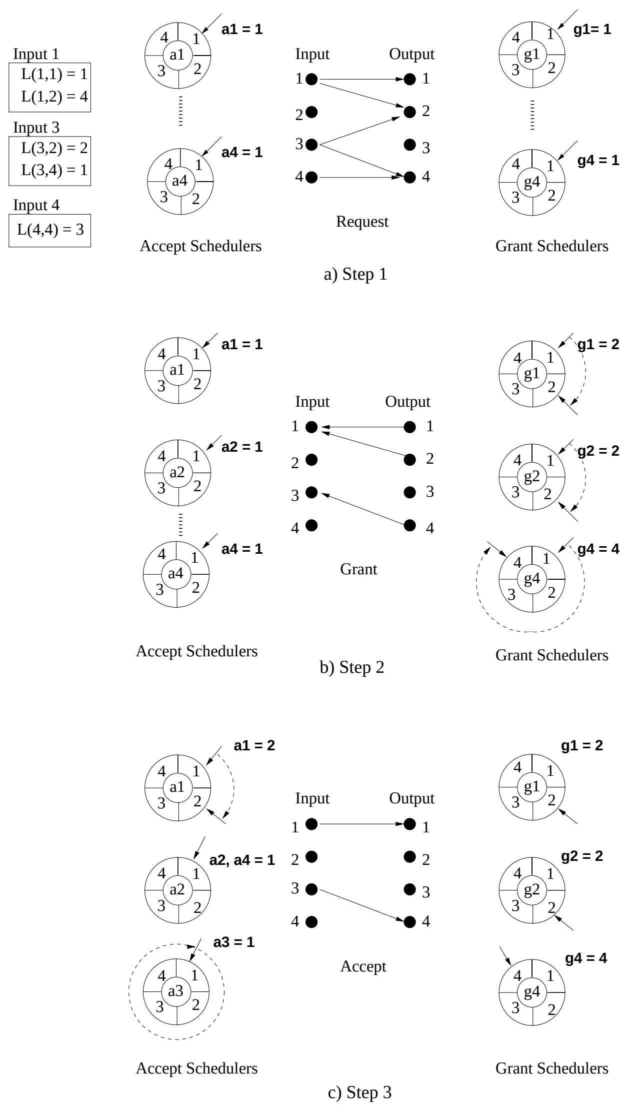

Fig. 4. Example of matching with iRRM

An example of this matching scheme is shown in Figure 4. In this example, we assume that the initial values for the grant pointers are input 1 (e.g., $g_i = 1$). Similarly, accept pointers are pointing initially to output 1 (e.g., $a_j = 1$). This status remains during step 1, while the inputs request transmission to all outputs that they have a cell destined for. In step 2, among all received requests, the grant schedulers select the requesting input that is nearest to the one currently pointed to. Output 1 chooses input 1, output 2 chooses input 1, output 3 has no requests, and output 4 chooses input 3. Then, the pointers g_i move one position beyond the selected one, in this case, $g_1 = 2$, $g_2 = 2$, $g_3 = 1$, and $g_4 = 4$. In step 3, the accept pointers decide which grant is accepted in similar way to the grant pointers. In this example, input 1 accepts output 1, and input 3 accepts output 4, then $a_1 = 2$, $a_2 = 1$, $a_3 = 1$, and $a_4 = 1$. Notice that the pointer a_3 accepted the grant issued by output 4, so the pointer returns to position 1.

IV. Iterative Round-Robin with Slip (iSLIP)

This scheme presented in [9], is an improved version of simple round-robin matching. The difference is that in this scheme, the grant pointers update their positions only if their grants are accepted (recall that there are two pointers, a grant pointer at an output port and an accept pointer at an input port). In this scheme, starvation is avoided because a recently matched pair gets the lowest priority. This scheme converges in, at most, N iterations. Although, one iteration is sufficient to offer 100% throughput. The matching procedure with this scheme follows three steps:

1) Each unmatched input sends a request to every output for which it has a queued cell.
2) If an unmatched output receives multiple requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer g_i to the highest priority element of the round-robin schedule is incremented (module N) to one location beyond the granted input *if and only if* the grant is accepted in step 3 of the first iteration. *The pointers g_i are only updated in the first iteration.*
3) If an input receives multiple grants, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer a_j to the highest priority element of the round-robin element is incremented (modulo N) to one location beyond the accepted output. As the granted pointers, *the pointers a_i are only updated in the first iteration.*

Because of the round-robin moving of the pointers, we can also expect the algorithm to provide a fair allocation of bandwidth among all flows.

This scheme contains 2N arbiters, where each arbiter is implementable with low complexity [6]; however, simulations were done with only one iteration due to the speed limitation of actual arbiters. The throughput offered with this algorithm is 100%, for any number of iterations. A matching example of this scheme is shown in Figure 5. Considering the example from the iRRM discussion, initially all pointers a_j and g_i are set to 1. In step 2 of iSLIP, the output accepts the request that is closer to the pointed input in a clockwise direction; however, in a manner different from iRRM, *the pointers g_i are not updated in this step. They wait for the acceptance result*. In step 3, the inputs accept the grant that is closer to the one pointed to by a_i. The accept pointers change to one position beyond the accepted one, $a_1 = 2$, $a_2 = 1$, $a_3 = 1$, and $a_4 = 1$. Then, after the accept pointers decide which grant is accepted, the grant pointers change to one position beyond the accepted grant (i.e., a non-accepted grant produces no change in a grant pointer position). The new values for these pointers are $g_1 = 2$, $g_2 = 1$, $g_3 = 4$ and $g_4 = 1$. In the following iterations, in the current arbitration phase, the pointers are not modified (i.e., updating occurs in the first iteration only). Only the unmatched input and outputs are considered in subsequent iterations.

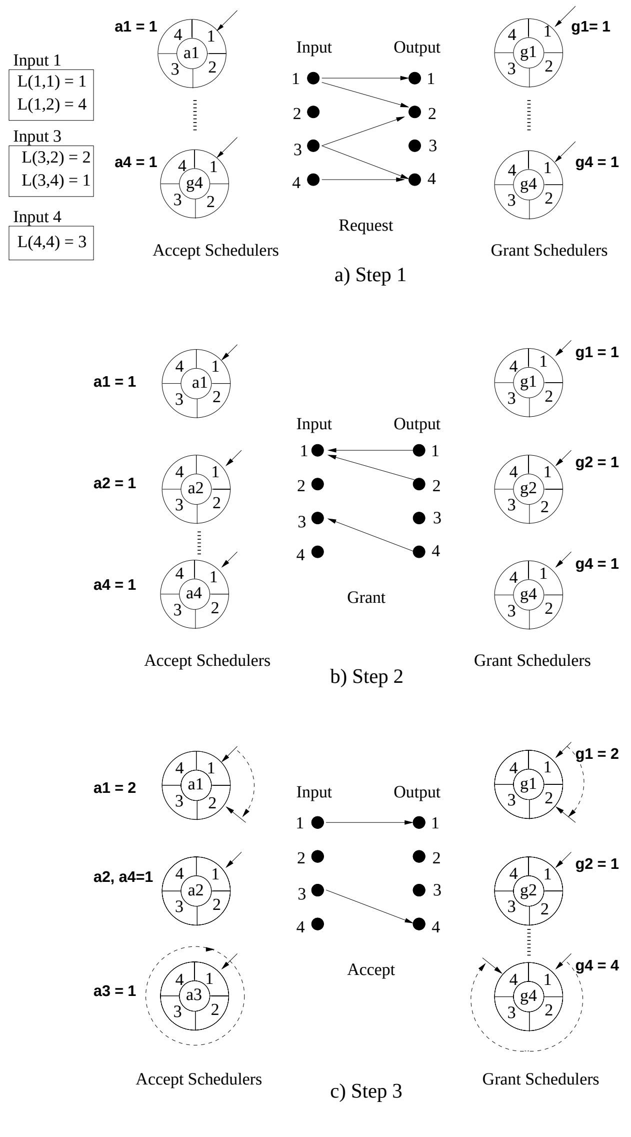

Fig. 5. Example of matching with iSLIP

V. Iterative Oldest Cell First (iOCF)

This algorithm, as in [7], considered for an IQ architecture, gives preference to cells that have been waiting for the longest time. This algorithm is reported to achieve 100% throughput and non-input starvation. It uses VOQ and no speedup ($S = 1$). It is stable for all independent arrival processes, i.e., no input queue occupancy grow infinitely. This is the maximal matching version of OCF. This scheme works in an iterative way, similar to iSLIP, except that iOCF uses a maximal-weight matching scheme. In this case, every request has a different weight (contrary to iSLIP, which uses equal request weight) equivalent to the queuing time of the HOL cell in the input queue. This algorithm is reported to need $\log_2 N$ iterations. It follows three different steps:

1) Request: each input sends a request to all outputs for which it has a queued cell. Each request contains the age of the first cell in the corresponding input queue.
2) Grant: When an output receives requests from more than one input, the request from the input with the oldest cell is accepted. In this description, ties are resolved by a random choice.
3) Acceptance: If an input receives two or more grants from more the output arbiters, the grant for the oldest cell is selected. Again, ties are resolved by a random choice.

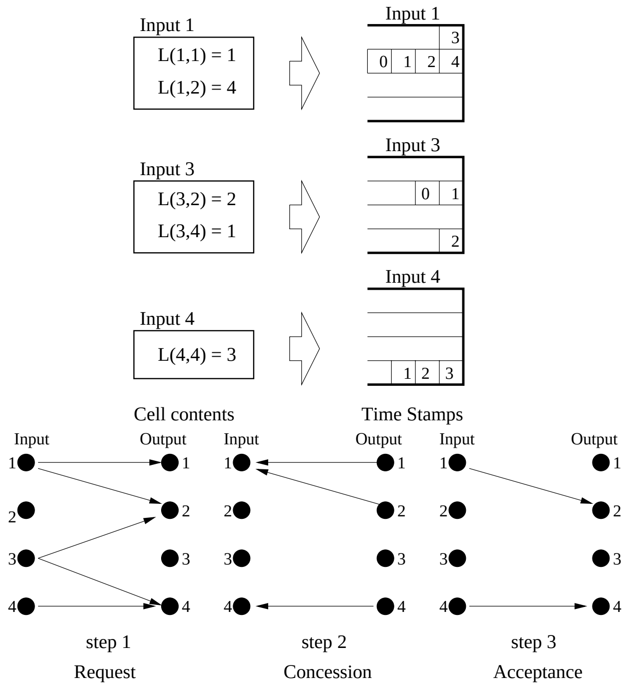

Fig. 6. Example of matching with iOCF

This scheme can achieve 100% throughput for general independent arrival processes with uniform distributed destinations. The implementation complexity is compromised because the age of the cells has to be stored. An example is shown in Figure 6. In this case, $TS(c)$ is the number of time slots that these cells have been waiting. In step 1, the request stage, every input sends requests to all outputs for which it has a cell destined. In step 2, the concession stage, outputs with multiple requests select the cell with the longest waiting time; ties are broken randomly. In this example, output 1 receives a unique request, output 2 selects the input 1 request due to the longest waiting time against input 3. Output 4 selects input 4. In step 3, the acceptance stage, inputs with multiple requests accept the grant that belongs to the cell with the longest waiting time or the oldest cell. Once again, ties are broken randomly. Here, only input 1 receives two concessions. Then, it selects the one belonging to output 2, due to cell age. In [18], this scheme was used with a speedup larger than 2 ($S > 2$), where the architecture was a CIOQ with rate-controlled schedulers at the input ports, giving not only 100% throughput but also a delay bound.

VI. Crossbar and Scheduler Implementations

The design and implementation of a cell-based crossbar follows a synchronous logic design. The crossbar has the following inputs: a master clock, a global reset, eight data inputs with a width of 8 bits each, and eight data outputs. In addition, the crossbar has an internal interface for the scheduler, which controls each one of the 64 crosspoints. Every crosspoint corresponding to input i receives the same input data at once. According to the scheduler results, a crosspoint let the data flow to the output in a synchronous way. When a crosspoint permits the flow of data to an output, it is said that this crosspoint is in close state, otherwise, it is said to be in open state. A crosspoint state lasts for the duration of the transmission of a cell. This duration is called a time slot.

The crossbar has a fault detection unit that detects when two or more crosspoints are in close status (invalid modes) and clears the data output to avoid sending corrupted cells to the output ports.

The crossbar provides the interface between the input ports and the scheduler. Therefore, communication between these two is pursued as in-band fashion. Every input has a cell-header manager unit. Figure 8 shows the logic of the header manager. The cell-header manager

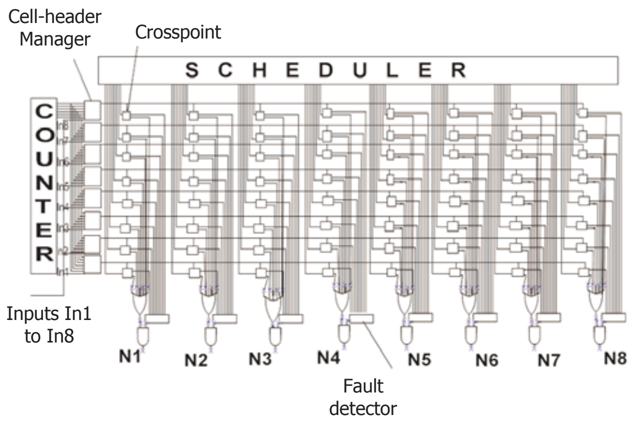

Fig. 7. Logic of an 8x8 crossbar

unit takes the request send by the input ports to the scheduler and exchange them for acceptance information issued by the scheduler. This information is needed by the input ports to determine the cells that can be transmitted through the crossbar.

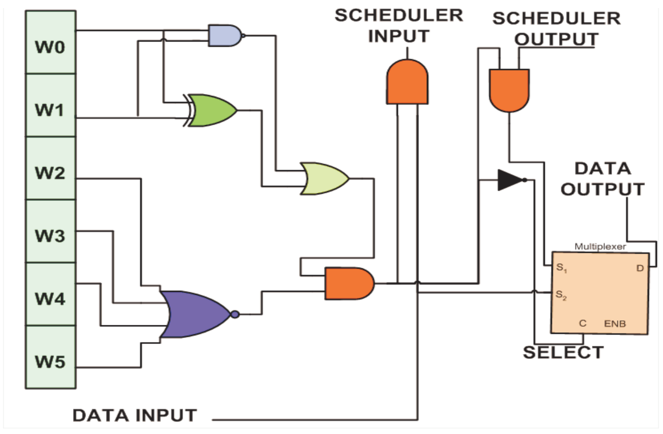

Fig. 8. Logic of the header manager section

The scheduler has 16 arbiters in a 8×8 switch. One per input/output. An output arbiter received the requests from the input ports and selects one. Then, an output arbiter informs every input arbiter about whether their requests are granted or not. An input arbiter selects a grant among all those received, and informs the crosspoint, input ports, and output arbiters about its decision. The crosspoint set its state as open or close for the next cell slot. The input ports take this information to determine which cell can be dispatched in the next time slot. The output arbiters take the acceptance information to update their pointers. Figure 9 shows the arbiters that form a scheduler section. The $Req(i,j)$ labels indicate the requests signals from input i to output j. The $Acc(i,j)$ signals indicate the acceptance information from the input arbiters at input i to output arbiters at output j.

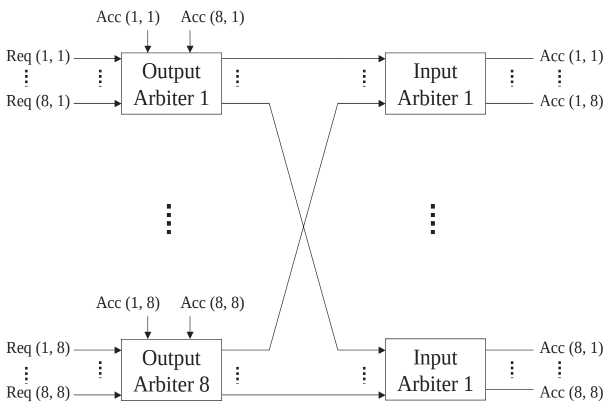

Fig. 9. Input and output arbiters in the scheduler

VII. Conclusions

This paper introduced the basis for switch fabric design. The switch fabric is a major component for a packet switch or Internet router. The performance of a packet switch is greatly determined by its architecture and, in our case, by the matching scheme used. The matching scheme is performed by the scheduler. The scheduler determines the time at which a cell (packet) can traverse the switch fabric in the trip to a destination. The objective of the scheduler is to resolve contention at the input and at the outputs. We described three different matching schemes. Two based on round-robin selection and one based on time stamps. These three schemes were implemented as a part of the undergraduate senior design project. The schemes were modelled with VHDL.

References

[1] M. Karol, M. Hluchyj, “Queuing in High-performance Packet-switching,” *IEEE J. Select. Area Commun.*, vol. 6, pp. 1587-1597, December 1988.

[2] M. Karol, M. Hluchyj, S. Morgan, “Input versus output queuing on a space-division switch”, *IEEE Transactions on Communications*, vol. 35, pp. 1347-1356, December 1987.

[3] A. Mekkitikul, N. McKeown, “A practical scheduling algorithm to achieve 100% throughput in input-queued Switches”, *Technical Report, Stanford University*.

[4] N. McKeown, A. Mekkittikul, V. Anantharam, J. Walrand, “Achieving 100% Throughput in an Input-queued Switch,” *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1260-1267, August 1999.

[5] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Tacker, “High-speed Switch Scheduling for Local Area Networks,” ACM Trans. on Computer Systems, vol. 11, no. 4, pp. 319-352, November 1993.

[6] I. Iliadis, W. E. Denzel, “Performance of packet switches with input and output queueing”, *Proc. ICC’90*, Atlanta, GA., April 1990. pp. 747-53.

[7] N. McKeown, “Scheduling algorithms for input-queued cell switches,” Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Univ. California at Berkeley, Berkeley, CA, 1995.

[8] N. McKeown, P. Varaiya, J. Warland, “Scheduling cells in an input-queued Switch”, *IEEE Electronics Letters*, pp. 2174-5, December 1993.

[9] N. McKeown, “The iSLIP scheduling algorithm for Input-queued Switches,” IEEE/ACM Trans. Networking, vol. 7, no. 4, pp. 188-201, April 1999.

[10] H.J. Chao, J-S. Park, “Centralized Contention Resolution Schemes for a large-capacity Optical ATM Switch,” IEEE ATM Workshop 1998, pp. 11-16, May 1998.

[11] Y. Li, S. Panwar, H.J. Chao, “The Dual Round-robin Matching Switch with Exhaustive Service,” IEEE HPSR 2002, pp. 58-63, May 2002.

[12] Y. Jiang and M. Hamdi, “A fully Desynchronized Round-robin Matching Scheduler for a VOQ Packet Switch Architecture,” IEEE HPSR 2001, pp. 407-411, May 2001.

[13] Y. Jiang and M. Hamdi, “A 2-stage Matching Scheduler for a VOQ Packet Switch Architecture,” IEEE ICC 2002, vol. 4, pp. 2105-2110, May 2002.

[14] D. N. Serpanos and P. I. Antoniadis, ”FIRM: A Class of Distributed Scheduling Algorithm for High-speed ATM Switch with Multiple Input Queues,” INFOCOM 2000, vol. 2, pp. 548-555, March 2000.

[15] C-S. Chang, D-S. Lee, and Y-S. Jou, “Load Balanced Birkhoff-von Newman Switches,” IEEE HPSR 2001, pp. 276-280, May 2001.

[16] A. Bianco, M. Franceschinis, S. Ghisolfi, A.M. Hill, E. Leonardi, F. Neri, R. Webb, “Frame-based Matching Algorithms for Input-queued Switches,” IEEE HPSR 2002, pp. 69-76, May 2002.

[17] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, “CIXB-1: Combined Input-One-cell-crosspoint Buffered Switch,” IEEE HPSR 2001, pp. 324-329, May 2001.

[18] A. Charny, “Providing QoS guarantees in input buffered cross-bar switches with speedup”, *PhD. Thesis, MIT*, 1998.