Design and Implementation of a Fast VOQ Scheduler for a Switch Fabric

A.H.Darvishan[†], H.Yeganeh[††], F.Sarabchi [†††]

Iran Telecommunication Research Center Tehran, Iran.

Abstract— The technical manufacturing, which is based on Switch Fabric and VOQ (Virtual Output Queuing), is used in speedy routers, which are used in core network. For configuring of Switch Fabric having speedy scheduler with high Throughput is required. The most promising algorithm is *i*SLIP which is an iterative algorithm that provides high efficiency for best-effort traffic. In this paper the way of designing and implementing of Prioritized *i*SLIP algorithm on FPGA are described. Because of the simplicity of this algorithm in implementing and needlessly of this algorithm to high rate switches (Tera bps), this algorithm is a proper method for Scheduling.

Keywords— Router, Switch Fabric, *i*SLIP, VOQ

I. INTRODUCTION

Over the past few years, it has been very difficult to accommodate explosively growing internet traffic demands by means of legacy routers. The networking industry was caught off-guard by this enormous rate of growth and has been unable to meet the increased demand for bandwidth. This is quite apparent to users connect to the internet at busy times of the day. In an attempt to keep up with demand, service providers are installing new links and upgrading switching and routing equipment. But it is now becoming apparent that the vendors of switches and routers are unable to meet current bandwidth demands, particularly at the core of the Internet, and are falling further and further behind. It is widely believed that this problem will be solved when more traffic is carried by high bandwidth ATM networks.

In the very simplest switch fabrics, all of the cells waiting at each input are stored in a single FIFO queue. When a cell reaches the head of its FIFO queue, it is considered by the centralized scheduler. For input-queued switches to be efficient, we must overcome the limitations of Head of Line (HOL) blocking [11], [13]. Many techniques have been suggested to reduce HOL blocking, although most are highly sensitive to traffic arrival patterns and perform no better than regular FIFO queuing for burst traffic. But HOL blocking can be eliminated entirely by using a simple buffering strategy at each input port, which is called "Virtual Output Queuing" (VOQ); each input maintains a separate FIFO queue for each
output [7].

When VOQ is used, the switch requires a scheduling algorithm that examines the contents of the N*2 (N is number of input-output) input-queues at the beginning of each cell time, deciding which ones will be served. It is shown [12] that a switch that uses VOQ can theoretically achieve 100% throughput for uniform or non-uniform arrival patterns. Unfortunately, these scheduling algorithms have typically been slow, inefficient to implement [1], [2], [3] and [4]. The most promising algorithm is *i*SLIP [8] which is an iterative algorithm that provides high efficiency for best-effort traffic. One of the profits of this algorithm is simplicity of implementation in hardware [5], [6]. *i*SLIP achieves fairness using independent round-robin arbiters at each input and output. But with simple round-robin arbiters, many outputs may try and connect to the same input each cell time.

In this paper, we survey Prioritized *i*SLIP algorithm and its implementing on FPGA. The outline of this paper is as follows: firstly the system architecture of routers is described in section 2, and *i*SLIP algorithm is introduced in section 3. Then, we propose the process of implementing this algorithm on FPGA and show simulation results on section 4 and 5.

II. SYSTEM ARCHITECTURE

In fig 1 different parts of a router are shown. Line Interfaces provide a physical connection between data links and equipment. The Network Processor analyses the IP Header and by this way it can have a control on errors, packet routing and classification. The Switch Fabric switches the packets between inputs and outputs. In the other word after some processing on packet in line Interface and network processor and it determines the output line for sending the packet, this part performs the switching process [9].

The hardware's implementation of crossbar switches is simple and these switches can support high rate data transferring (for example Tera bps), so using the crossbar switch is a standard model for designing Switch Fabric.In fig. 2 the internal structure of a Switch Fabric is presented.

Manuscript received September 5, 2008.
Manuscript revised September 20, 2008.

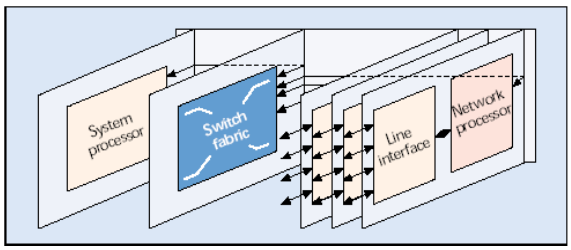

Fig 1: Router Architecture [9]

Switch Fabric has the below parts:

- ✓ ***CSIX Part*** includes a standard connector which connects the Switch Fabric to Network Processor and it implements the CSIX protocol. This standard defines the way in which the cells are transferred and also the flow control's information.
- ✓ ***IPP (Input Part Processor) and VOQ*** constitute the input part of the switch and it includes the requirement functionality for implementing the VOQ in order to prevent HOL Blocking problem, manage the buffers and support the flow control's process.[10]
- ✓ ***Output of the switch*** includes OPP (Output Port Processor) and OQ (Output Queue). In this part the cells are stored in CoS Queues. OPP part schedules the cells and sends them to Network Processor through CSIX by using WRR algorithm and Strict Priority. OPP is responsible for managing OQs and supporting the Information of flow control. The FCB (Flow Control Broadcast) block provides the information about OQs for all IPPs. If an OQ is occupied, related VOQs pause sending cells to OQ, after emptying the OQ, sending the cells is resumed. It must be said that Crossbar Section is common among Network Processors.

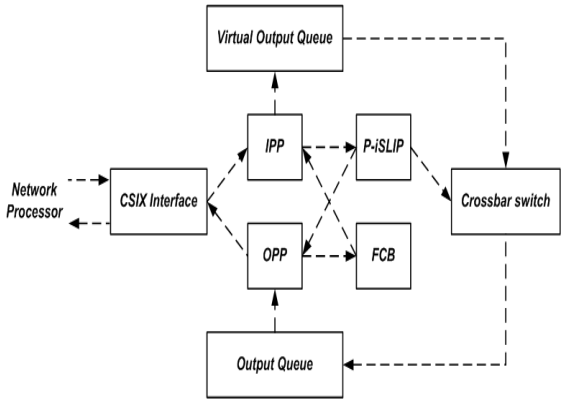

Fig 2: Switch Fabric Architecture

III. P-*i*SLIP ALGORITHM [8]

Many applications use multiple classes of traffic with different priority levels. The basic *i*SLIP algorithm can be extended to include requests at multiple priority levels with only a small performance and complexity penalty, which is called the Prioritized iSLIP algorithm.

In Prioritized iSLIP each input now maintains a separate FIFO for each priority level and for each output. This means that for an $N \times N$ switch with P priority levels, each input maintains $P \times N$ FIFOs. This algorithm has three steps which are described in below:

1- Request: Input selects the highest priority nonempty queue for output. The input sends the priority level of this queue to the output

2- Grant: An output accepts the request, which has the most priority, from an input in form of Round-Robin.

3- Accept: If an input receives any grants, it determines the highest level grant. The input then chooses one output among only those that have requested at this level, and then the above steps are repeated.

Implementation of the P-*i*SLIP algorithm is more complex than the basic algorithm, but can still be fabricated from the same number of arbiters.

IV. IMPLEMENTING P-*i*SLIP ALGORITHM

The Scheduler bock diagram is shown in fig.3. In each matrix the rows present the inputs and the columns present the outputs. Rij represents the priority of input i for connecting to output j which it will be select one of the priority levels between 1 to P. If there is no request from input i for output j, the value zero is assigned in matrix. Each column of the Request matrix is related to the Grant in output part, so each output can accept the related input. It must be noticed that in result vector of each Grant block, only one array (accepted input) is upper than zero (this value represents the requesting priority) and the other arrays are zero.

The Grant matrix represents that each output accepts which input. In each column of this matrix only one array is upper than zero and the others are zero. For establishing a connection between inputs and outputs, each column of this matrix is connected to related Accept block. In output matrix of these blocks, only one array is more than zero and the other arrays are zero. If in Accept matrix Aij is 1, then the other arrays in row i and column j will be 0. Also in Request matrix the array in row i and column j will be 0 and the steps of the algorithm will be repeated. The number of repetition depends on the time that is necessary for performing the algorithm and also the length of the cells. (When a cell is sending, the algorithm must be implemented and the connection must be identified).

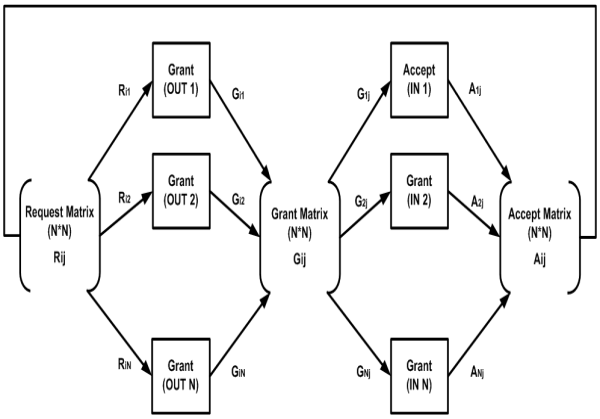

Fig 3: Internal structure of scheduler block based on P-*i*SLIP algorithm

According to the explanation of the algorithm, you will find that the second and the third steps are similar, but they perform on different information. So the internal structures of the Grant and Accept blocks are similar. The internal structure of these blocks, when there are four priorities for requests, is based on the fig 4.

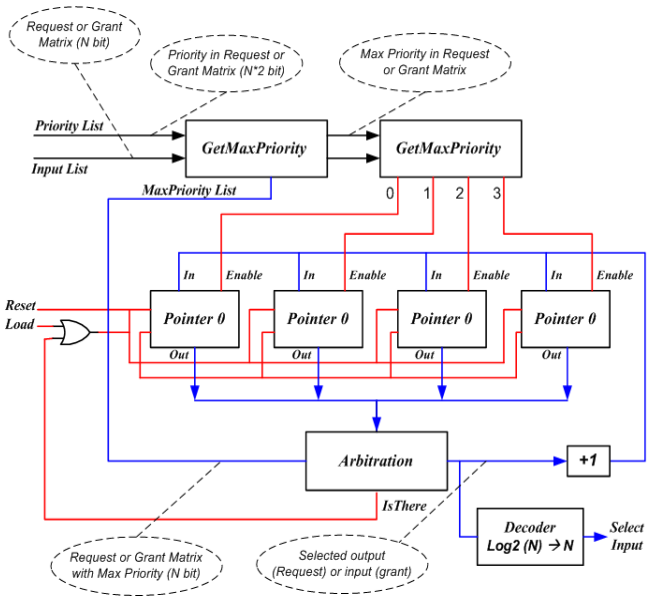

Fig 4: Internal structure of Grant and Accept

If we assume there are four priorities in designing the Scheduler, so there will be four pointers, each pointer is related to one priority level and it includes the highest priority of selected input. Implementation of these pointers is based on Latch with TriState output. The inputs of Arbiter are explained in below:

✓***Reset***: Which is used for resetting the pointers (It is used when the system is operated for the first time or is restarted).

✓***Load***: In each execution of P-*i*SLIP algorithm, it is calculated with Load signal then it will be put in related pointer (The pointer with the most priority).

✓***Priority List***: Each list includes N*2 bits (N is number of input-output), each two bits represents the priority level in selected request.

✓***Input List***: This list includes N bits. Each bit is related to one input (in Grant block) or one output (in Accept block). When the related input has no request or the related output doesn't give any Grant to related input, the zero value is assigned. It must be noticed that in this algorithm two matrixes are defined. One matrix represents the number of priorities and the other represents the existence of the requests.

The outputs of the blocks are:

✓***Select Input***: It includes N bits. Each bit is related to one input (in Grant block) or one output (in Accept block). It must be noticed just one bit is 1 and the other bits are 0. For each output only one Grant input is defined and in a similar way, for each input only one Accept output is defined.

Now the manner of working the circuit which is shown in fig 4 is explained. In the first step the highest priority of available requests is calculated (by the GetMaxPriority block). This maximum will activate the pointer of this priority level by a decoder. If no requests exist (Input List completely null); the Arbitration block set the "isThere" signal to 1 and this lets the existing value of the pointer not to change by activation of the Load signal. It should be noted that the pointers have TriState output and they are connected to each other directly by these outputs. By activation of the related pointer and based on its value, the Arbitration block, select on of the inputs from the list (Requests or Grants) by using the P-*i*SLIP algorithm, and put it in its output. On the one hand this output will connect to a decoder and the decoder's output will be used for connecting to other Scheduler blocks (This decoder is used for simplifying of implementation other blocks) and on the other hand, the value of Arbitration output will be added with one and it will be saved in the activated pointer by the arrival of Load signal from Controller Block. It should be noted that for the execution of the next P-*i*SLIP, the value of the pointer should be updated by the number after the accepted input.

The function of the GetMaxPriority is to find maximum priority in the list of the arrived requests. By finding this maximum, the cases which have the maximum priority will be selected from the list of input requests and will be set in the output of this block. This output will go to the Arbitration block, and a request will be accepted or granted.

Implementation of this block is shown in figure 5, which is used to find maximum priority in a list includes N number with 2 bits (because we have 4 priorities in this example). This numbers can choose on of the {00, 01, 10, 11} digits. The proposed Circuit specifies whether there are any 10 or 11 between these numbers by computing OR function over the entire of the most significant bits of these N numbers. Actually if the result is 1, then it means N includes one of these numbers (01 or 11). Thereupon the most significant bit of output (MaxPriority) set to 1. If the OR result is zero, then it means there are only 00 or 01 in this list, hence the most significant bit of the MaxPriority set to 0. Anyway, it's noticed that the output of OR function is the equal to the most significant bit of the MaxPriority output. But the least significant bit has more ambiguity. For obtaining the least significant bit MaxPriority, computes OR function over the entire of the least significant bits of N numbers (Temp1). Therefore we can find if the list includes 00 or 10, the result of OR function will be 0, and if the list includes 01 or 11, the result of OR function will be 1. The ambiguity which is mentioned occurs when 00, 01, and 10 are in the list. In this case the output of proposed circuit is 11, which is wrong prediction. Hence, if we want to specify the least significant bit of MaxPriority we must perfect the circuit. Therefore, we utilize Temp0, Temp1, and Temp 2.

Moreover, the GetMaxPriority block has another task in the proposed system, which is used for distinguishing the existence of requests. In this case the input of GetMaxPriority block is an N-bits string that presents existence or non-existence of a request. If in this string a bit is zero, it means that for that number (An input that should be granted or an output that should be accepted) there is not any request. If so,

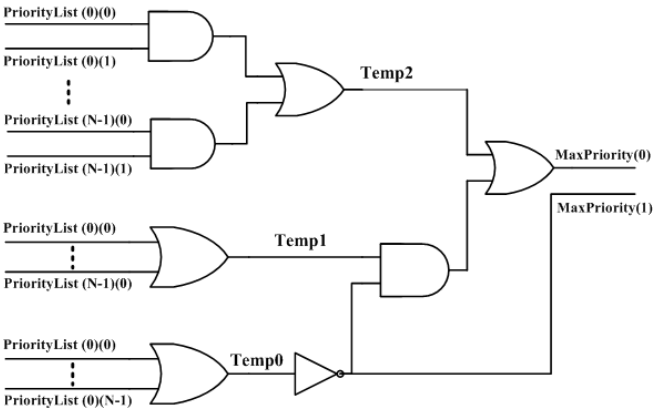

Fig 5: GetMaxPriority Implementation

in PriorityList in the same position, 00 should be set as the priority; hence it won't include in the maximum priority calculation. This Restriction should be applied from previous levels because this will lead to the elimination of a level of gates in implementation.

After calculating the maximum priority, the requests whit a maximum priority should be answered. Actually, output of this block is an N- bit number that each bit indicates the state of related request. So Arbitration block's input is requests with maximum priority. Circuit of this part of the GetMaxPriority block is shown in figure 6. It should be noted that this process repeats for each array of MaxPriority list (i=0… N-1). XOR gates compare PriorityList arrays with MaxPriority arrays. AND gate is used for applying existence or nonexistence of input requests.

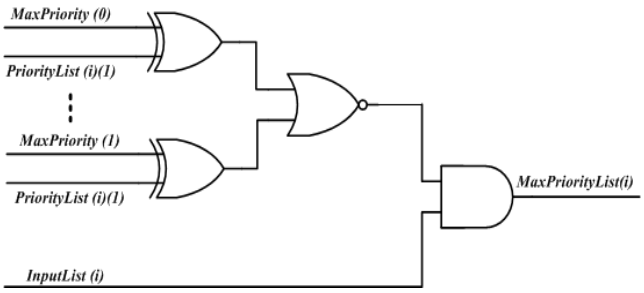

Fig 6: Input with Maximum Priority

After specifying request with highest priority and the value of the pointer, the Arbitration block selects the request with maximum priority based on the P-*i*SLIP algorithm and sends it to the output. If there are no requests, “isThere” output set to 1. It causes the Load input which comes from Controller block don’t change the state of pointers.

In figure 7 the VHDL simulation result of proposed system is shown for various values. In the first example, value of the pointer is zero and it means that the port number zero has the maximum priority. As in the list of requests, this port is applicant, so, based on P-*i*SLIP it will be selected. In the second example the pointer's value points to port number 2 but as the port 4 is selected in the list, this port is assigned to the output. In the next example, port 6 has maximum priority but based on the rotating system in P-*i*SLIP port 2 will be selected. In the next example, there is no request and so the “isThere” output will be activated, therefore the value of the pointers doesn’t change by activation of the load signal. In this case, the value of output will be valueless. At the end in the last example, the port 5 is pointed but port 7 will be selected.

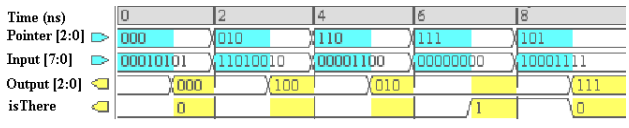

Fig 7: Simulation of Arbitration Block

V. Simulation and Conclusion

Aforesaid plan has been used in the MPLS switch/router project in Iran Telecommunication Research Center and the proposed system is simulated and verified by VHDL. Figure 8 shows the percentages of connections are established as a function of algorithm iteration. In this simulation the algorithm is evaluated by 10000 random request matrixes and for different number of ports (N=8, 16, 32) and different priority level (2 and 4) the number of established connections is calculated for each iteration.

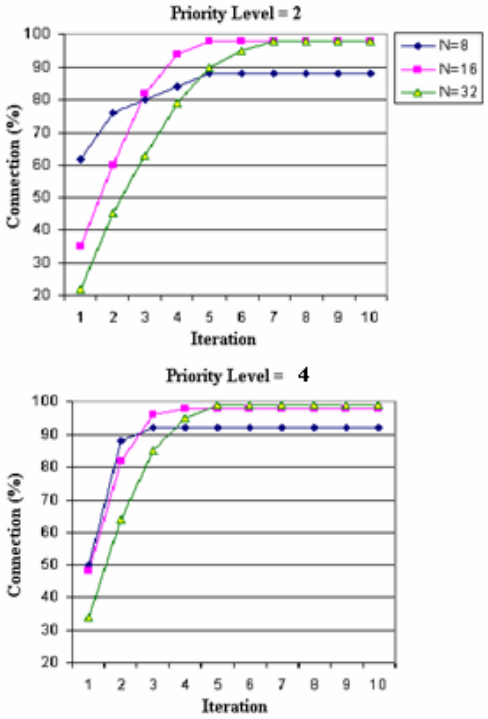

Fig 8: Simulation Results

As it's seen, by increasing the number of priority levels and the number of ports the efficiency of the algorithm will be improved but more hardware resources and execution time will be needed. Also by increasing the number of iteration, connection which is established is increased, but from one point it tangents to the maximum available connection, so from this point increasing the iteration number will not be effective. On the other hand, it's seen that increment of priority level is afforded earlier saturation of connection percentage.

In the synthesis is made for 16×16 switch with 8 times algorithm iteration, the delay of scheduler will be about 60ns that is suitable in construction of a rapid router.

References

[1] Anderson, T.; Owicki, S.; Saxe, J.; and Thacker, C. "*High speed switch scheduling for local area networks*," ACM Trans. on Computer Systems. Nov 1993 pp. 319-352.

[2] Karol, M.; Eng, K.; Obara, H. "*Improving the performance of input-queued ATM packet switches*," Proc. of IEEE INFOCOM '92, pp.110-115.

[3] Obara, H. "*Optimum architecture for input-queueing ATM switches*," IEE Electronics Letters, pp.555-557, 28th March 1991.

[4] Obara, H.; Hamazumi, Y. "*Parallel contention resolution control for input-queueing ATM switches*," *IEE Electronics Letters*, Vol.28, No.9, pp.838-839, 23rd April 1992.

[5] N. McKeown; P. Varaiya; and Walrand, Jen; "*Scheduling Cells in an Input-Queued Switch*," IEE Electronics Letters, Dec 9th 1993, pp.2174-5.

[6] McKeown, Nick; "*Scheduling Cells in Input-Queued Cell Switches*," PhD. Thesis, University of California, Berkeley, 1995.

[7] N. McKeown, M. Izzard, "*High Performance Switching*",

[8] N. McKeown, "*The iSLIP Scheduling Algorithm for Input-Queued Switches*", IEEE/ACM Transactions on Networking, 7(2):188-201, April 1999.

[9] Dr.Kamran Sayrafian , ZAGROS Networks Corp. "*Owerview of Switch Fabric Architecture*"

[10] P. Gupta and N. McKeown, "*Design and Implementation of Fast Crossbar Scheduler*," Hot Interconnects VI, Stanford University, Aug 1998.

[11] M.Karol, M. Hiluchyj,and S.Morgan, "*Input Versus Output Queueing on a Space Division Switch*", IEEE Trans. on Comm., vol. 35, no.12, pp. 1260-1267, 1987.

[12] N. McKeown, V. Anantharam, and J. Walrand, "*Achieving 100% throughput in an input-queued switch*," in Proc. IEEE INFOCOM '96, San Francisco, CA, pp. 296–302.

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