A Reconfigurable Hardware Based Embedded Scheduler for Buffered Crossbar Switches

Lotfi Mhamdi
lotfi@ce.et.tudelft.nl

Christopher Kachris
kachris@ce.et.tudelft.nl

Stamatis Vassiliadis
S.Vassiliadis@ewi.tudelft.nl

Computer Engineering Laboratory
EEMCS, TU Delft
P.O. Box 5031, 2600 GA Delft, The Netherlands

ABSTRACT

In this paper, we propose a new internally buffered crossbar (IBC) switching architecture where the input and output distributed schedulers are embedded inside the crossbar fabric chip. As opposed to previous designs, where these schedulers are spread across input and output line cards, our design allows the schedulers to have cheap and fast access to the internal buffers, optimizes the flow control mechanism and makes the IBC more scalable. We employed the Xilinx Virtex-4FX platform to show the feasibility of our proposal and implemented a reconfigurable hardware based IBC switch with the maximum port count that we could fit on a single chip. The experiments suggest that a 24×24 IBC switch running a 10 *Gbps* port speed and a clock cycle time of 6.4 ns can be implemented.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communications**]: Interconnections (Subsystems); B.6.3 [**Logic Design**]: Design Aids; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design, Performance

Keywords

Scheduling, Buffered Crossbar Fabric, Reconfigurable Hardware

1. INTRODUCTION

As input queued (IQ) crossbar switches reach their practical limitations due to higher port numbers and data rates, internally buffered crossbar (IBC) switches are gaining a lot of interest due to their great potential in solving the complexity and scalability issues faced by their buffer-less predecessors [3]. As opposed to traditional IQ switching, where a centralized and complex scheduler is required [8], for an $N \times N$ IBC switch, there are N input schedulers and N output schedulers. These schedulers are decoupled and can work independently in parallel. A scheduling cycle consists of three independent phases: input scheduling, output scheduling and flow control mechanism. The flow control informs the input (respectively output) schedulers about the status (occupancy) of the internal buffers. It is the only communication means throughout which the schedulers communicate in order to perform their arbitrations and prevent internal buffers overflow.

A plethora of scheduling algorithms has been proposed for the IBC switching architecture [9, 11, 10]. The vast majority of these algorithms have been designed under the assumption that the input schedulers are located in the input line cards -one per each card- and the output schedulers are placed at the output side. This implies that, every time slot, the flow control mechanism has to communicate to every input (respectively output) scheduler the occupancy of its corresponding internal buffers. This can be considered not only as costly in terms of latency and I/O pins but also a scalability limiting factor.

In this paper, we propose a novel design for the IBC switching architecture where the input and output schedulers are all embedded within the crossbar fabric chip. The idea stems from the fact that the crossbar fabric switch is bound by pin count and not by the memory amount inside the chip. VLSI density increases [14] make it possible to include enough memory inside the crossbar fabric chip. The fabric I/O pin count constraint implies that there must be unused area inside the chip that can be more effectively used. The benefits of our proposed design are:

- Optimizing the flow control mechanism between the crossbar fabric chip and the schedulers. This has the benefit of speeding up the scheduling time while using a limited number of I/O pins resulting in more scalable IBC crossbar switches.

- Improving the performance of the scheduling algorithms, as there are many algorithms that are basing their decisions on the internal buffers and when embedded within the crossbar chip would have faster decisions and cheaper access to resources.

- More effective use of the crossbar chip area and saving area on the input (respectively output) line cards that could be used for additional tasks.

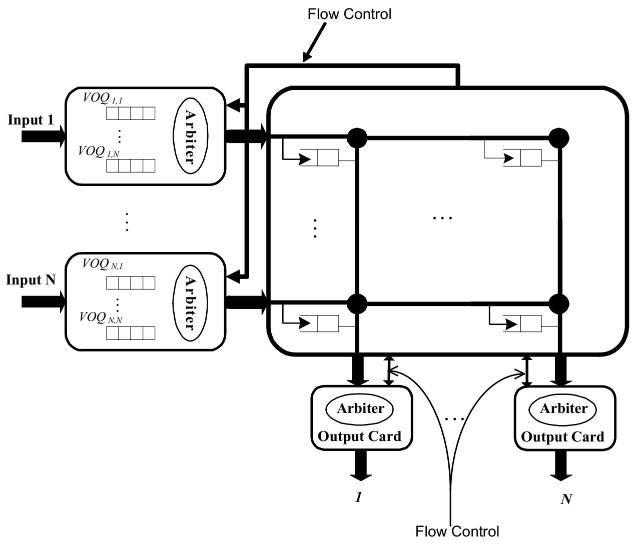


Figure 1: The IBC Switching Architecture.

In particular, we showed the feasibility of such design for 24×24 IBC switch with a 10 *Gbps* port speed. The target Technology was the Xilinx Virtex-4FX [12]. Our design can be used to implement broader classes of scheduling algorithms such as LQF-RR [6] at no extra cost.

The remainder of this paper is organized as follows: Section 2 presents background knowledge. Section 3 introduces the IBC architecture along with embedded schedulers. We present a possible hardware implementation of a scheduling algorithm. Section 4 contains implementation results and performance evaluation. Finally, Section 5 gives some concluding remarks.

2. BACKGROUND KNOWLEDGE

Buffered crossbar switches have been studied since almost two decades and different types of architectures have been proposed [1, 10]. The most widely used architecture is the one based on virtual output queuing (VOQ) as depicted in Figure 1, and was first proposed by [10]. The VOQ technique is used to overcome the Head-of-Line (HoL) blocking phenomenon [8]. Numerous researchers have been working on the design and implementation of IBC switches [7, 14]. For an $N \times N$ IBC switch, there are N input line cards, N output cards and buffered crossbar fabric. Each input line card consists of N logically separated VOQs (one per output) and an input scheduler. An output card contains a memory buffer and an output scheduler. The buffered crossbar fabric is simply a crossbar fabric in which there exists a limited amount of memory per cross point. During the course of their operations, the input schedulers as well as the output schedulers critically rely on a flow control mechanism. The flow control mechanism is a crucial component due to its effect on the scheduler's overall performance and the cost and implementation feasibility of the crossbar chip.

A broad class of scheduling algorithms has been proposed for the IBC switching architecture. These algorithms can be classified into round robin based algorithms [11] and weighted algorithms [9, 6] or a combination of the two. Most of the proposed algorithms have been designed with the assumption that the input schedulers are taking place at the input line cards and the output schedulers are placed on the output cards. When the input schedulers are implemented on the input line cards, the flow control mechanism can be the bottleneck as the number of ports of the switch increases or the speed per port increases. For an $N \times N$ IBC switch, every time slot the flow control mechanism has to carry N bits (one per each cross point buffer) to each input scheduler in order for the latter to know which internal buffer to be served next. Clearly, as N increases, the crossbar implementation becomes infeasible due to I/O pin limitation. The alternative solution to this problem is to sacrifice time instead of pins by using the same limited number of I/O control pins for all input schedulers over many time slots, resulting in longer arbitration times [14].

In our previous work, we were among the first who proposed scheduling algorithms that perform their decisions exclusively on the internal buffers only. We showed the importance of the internal buffer information in the arbitration process. Because of its shared nature between the input and output schedulers, the internal buffers occupancies can serve as a good arbitration criterion. In particular, our previously proposed MCBF (Most Critical internal Buffer First) algorithm [9] was shown to outperform all existing algorithms while basing its scheduling decision exclusively on the internal buffers occupancies only. In this paper, we further show that the input and output schedulers can all be embedded within the crossbar fabric chip. In particular, we show the implementation feasibility of such design for 24×24 IBC switch running at 10 *Gbps* per port. To the best of our knowledge, there has been no other study showing the feasibility of such embedded design. When embedding the scheduler within the crossbar fabric, the flow control mechanism is optimized resulting in the feasibility of implementing scalable switches both in terms of port numbers and speed per port.

3. EMBEDDED SCHEDULING ARCHITECTURE

In this section, we describe the proposed IBC switching architecture where the input and output schedulers are embedded within the crossbar fabric. To show the feasibility of our design, the MCBF algorithm [9] is implemented. For the sake of clarity, we introduce some notations that will be used throughout the remainder of this article.

3.1 Notation

We consider the switch model defined in Figure 2. Fixed size packets, or cells, are considered. Variable length packets are segmented into cells for internal processing and reassembled before they leave the switch. There are N input cards; each maintaining N logically separated $VOQs$. When a packet (cell), destined to output j, $1 \leq j \leq N$, arrives to the input card i, $1 \leq i \leq N$, it is held in $VOQ_{i,j}$. In addition to the above, we define the followings:

- Eligible VOQ ($EVOQ$): A $VOQ_{i,j}$, is said to be eligible (denoted $EVOQ_{i,j}$) for being scheduled in the input scheduling process if it is not empty and the internal buffer $XP_{i,j}$ is empty (or not full).
- The internal fabric consists of N^2 buffered crosspoints (XP), N input schedulers (IS) and N output scedulers

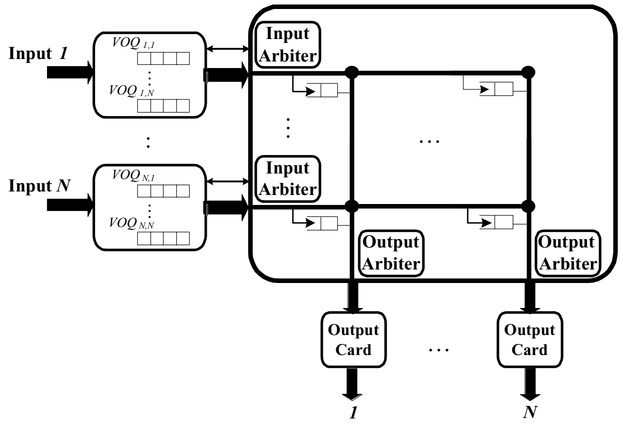


Figure 2: The IBC switching Architecture with embedded schedulers.

(OS). A crosspoint $XP_{i,j}$, holds cells coming from input i and going to output j.

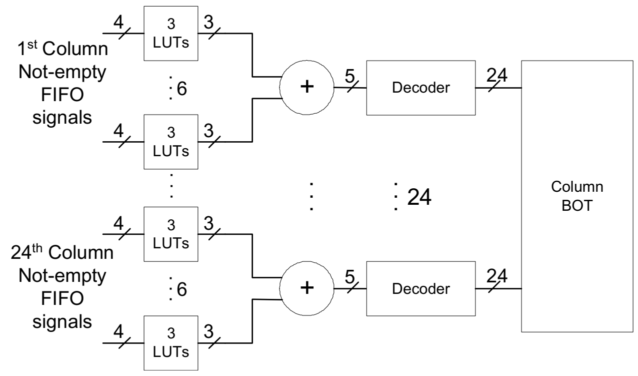
- The line of crosspoint buffers $LXPB_i$ is the set of all the internal cross point buffers ($XP_{i,j}$) that correspond to the same input, i, and holding cells for all outputs. NLB_i is the number of cells held in $LXPB_i$. IS_i schedules the arrival of cells from input card, i, to $LXPB_i$.
- The column of the cross point buffers $CXPB_j$ is the set of the internal buffers ($XP_{i,j}$) that correspond to the same output, j, and receiving cells from all inputs. NCB_j represents the number of cells queued in $CXPB_j$. OS_j arbitrates the departure of cells from $CXPB_j$.

Figure 2 depicts the interaction between the input line cards and the buffered crossbar fabric. Each input card uses 6 signals (5 signals for the 24 input queues and 1 signal for valid data) to notify the switch fabric card that a new packet has arrived to the VOQs (indexed by the 5 bits signal). Once the input scheduler has decided which input VOQ is selected, a 6 bits signal is sent back to the input line card containing the selected VOQ index. The cell is then forwarded to the corresponding internal buffer. Cells can be sent using the SERDES transceivers [13]. The output card is simpler than the input card. Cells are, again, sent using the SERDES transceivers as soon as the output scheduler makes a decision. Furthermore, no flow control is needed since the output scheduler is moved inside the buffered crossbar fabric chip.

3.2 MCBF Implementation

The MCBF scheduling scheme was proposed in [9]. It has the following specification:

- Input Scheduling (IS)
  - For each input i: Starting from the highest priority pointer's location, select the first $EVOQ$ corresponding to: $\min_j\{NCB_j\}$ and send its HoL cell to the internal cross point buffer ($XP_{i,j}$). Move the highest priority pointer to the location $(j+1)(mod\ N)$.



1st Column Not-empty FIFO signals
24th Column Not-empty FIFO signals
4
3 LUTs
3
6
24
5
Decoder
+
Column BOT

Figure 3: The Buffers Occupancy Table Controller.

Table 1: Encoding of the number of '1's

Not-Empty	LUT Output
0000	000
0001	001
0010	001
0011	010
0100	001
...	...
1111	100

- Output scheduling (OS)
  - For each output j: Starting from the highest priority pointer's location, select the first $XP_{i,j}$ corresponding to: $\max_i\{NLB_i\}$ and send its HoL cell to the output. Move the highest priority pointer to the location $(i+1)(mod\ N)$.

To efficiently map the input scheduler into hardware the following structure has been used. Inside the crossbar fabric, there is one 24×24-bit array, named Column Buffer Occupancy Table (C_BOT), and each row represents the number of occupied internal buffers, XP, for each column of the crossbar fabric. Each row is initialized with the first bit asserted "1" and all the others with "0". The position of the "1" in the row represents the number of occupied internal buffers in this column. The controller of the C_BOT is depicted in Figure 3. Each Xilinx Virtex Look-Up-Table consists of 4 inputs and 1 output. Hence, 4 not-empty signals are used as inputs to the LUTs to encode the number of ones, as it is shown in table 1. The number of the occupied buffers for a column are added and then decoded and forwarded to the C_BOT. For example, in figure 4, the first, the second and the fourth columns have 2 occupied buffers, while the third and the fifth columns have 3 occupied buffers.

The micro-architecture of the input arbiter is shown in figure 4. When a new packet arrives to the input card, a signal is asserted stating the id of the VOQ. The input arbiter first updates the Input Buffer Table (IBT). The IBT keeps the number of waiting cells in the input line card. The number of waiting cells is represented using 15 bits (up to 32 Kilo cells). The input card asserts a "new packet" for a specific VOQ only when the number of waiting cells in the corresponding entry in the table is less than 4 (The input

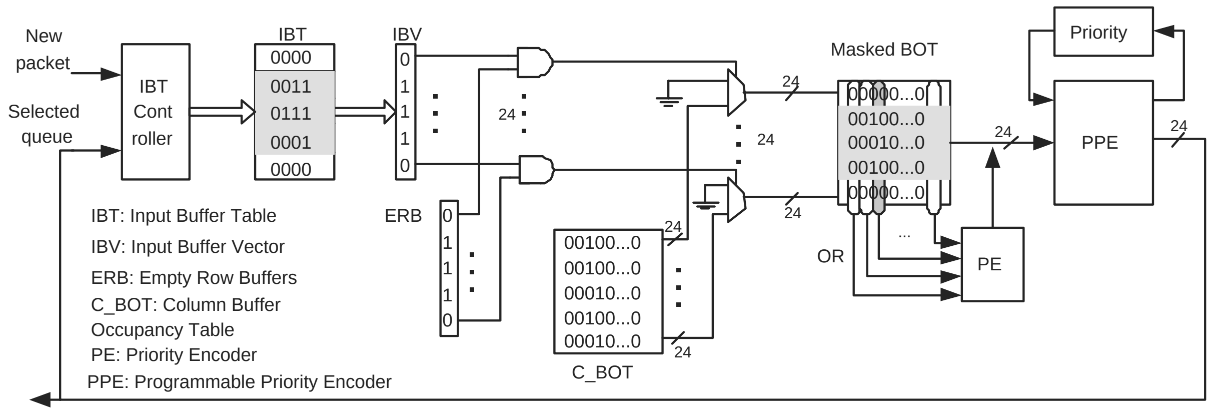


Figure 4: Input Arbiter Micro-Architecture.

card keeps a record of the number of new cells and selected cells of a VOQ). Thus, the IBT is used mainly to speedup the time consuming process of communication between the input card and the crossbar switch.

The Input Buffer Vector (IBV) is a 24-bit vector that represents the state of the IBT. If the row is 0 then the bit is also 0; otherwise it is 1. IBV can be presented as follows:

$$IBV_j = \begin{cases} 1 & \text{if } IBT_j = 0; \\ 0 & \text{otherwise.} \end{cases}$$

The IBV is AND-ed with the Empty Row Buffers (ERB) vector which represents whether the corresponding internal crosspoint buffer, XP, is free. The result is a vector that represents the eligible queues. This vector is used as a mask for the C_BOT and a new table (Masked BOT) is created that represents the number of occupied buffers of the eligible queues (EVOQs). Each row, j, of the Masked BOT (M_BOT_j) can be computed as follows:

$$M\_BOT_j = \begin{cases} C\_BOT_j & \text{if } IBV_j \ AND \ ERB_j = 1; \\ 0 & \text{otherwise.} \end{cases}$$

The elements of each column of this table are OR-ed and are forwarded to a priority encoder to find the eligible queues with the minimum number of occupied buffers. Using this micro-architecture we can easily locate the queue with the minimum occupied buffers of the eligible queues. Finally, this vector is forwarded to a programmable priority encoder to select the queue based on the round-robin based priority. The programmable priority encoder can be implemented in several ways as it is shown in [5]. In this implementation we used the faster one, segmented in 3 clock cycles.

The output arbiter, depicted in figure 5, is similar to, even simpler than, the input arbiter. A 24 × 24-bit array, named Row Buffer Occupancy Table (R_BOT), is used to store the occupancy of the internal buffers for each row of the buffered crossbar fabric. The position of the '1' in the entry, i, in the array represents the number of queued cells in the line of crosspoint buffers $LXPB_i$. For example, if the '1' is on the third position it means that $LXPB_i$ has 2 queued cells. The "non-empty column buffers" (NECB) is a 24-bit vector that represents the occupancy of the corresponding column of crosspoint buffers, $CXPB$, (if it is free or not) and it is used as a mask for the R_BOT to create a masked BOT. Each row, j, of the Masked BOT (M_BOT_j) can be computed as follows:

$$M\_BOT_j = \begin{cases} R\_BOT_j & \text{if } NECB_j = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Each 24-bit column of the masked BOT is OR-ed and the first column from the right with a non-zero value is forwarded to the Priority Encoder (PE). Then, this vector is forwarded to the Programmable Priority Encoder to decide which will be the selected crosspoint, XP, based on a round-robin priority scheme

3.3 Extension to wider range of Algorithms

Our design can be extended to implement a wider range of scheduling algorithms. For example, we can use our design and embed the Longest Queue First-Round Robin (LQF-RR) algorithm [6] or the Oldest Cell First (OCF-OCF) algorithm [10] in a similar manner as MCBF. We expect all these algorithms to be easily mapped inside the buffered crossbar chip because of their similar hardware requirements, as weighted schemes. In the case of the LQF-RR algorithm, the input scheduler (LQF) can be embedded within the buffered crossbar chip by using the IBT table with a slight modification. The LQF algorithm gives priority to the input VOQ with the highest number of packets (cells). As mentioned in the previous section, the IBT table uses 15 bits to represent the occupancy (length) of each input VOQ in number of cells. However, in practice, an input line card should hold up to 100 ms worth of packets [2]. At 10 *Gbps*, the buffer requirement per line card would be 125 MB. For a 24×24 IBC switch and 64 B cells, every VOQ is approximately 5.2 MB (or 81.25 K cells). This translates to an IBT entry of 17 bits, which can be easily modified. Besides the IBT modification, LQF would not require much of difference compared to the MCBF scheme. The same modification can be applied to the OCF algorithm in order to represent the arrival time of cells to the VOQs instead of queue length.

4. IMPLEMENTATION AND SIMULATION RESULTS

This section presents our hardware implementation results in terms of timing and area of our design. In addition to the implementation results, we also conduct a simulation study to evaluate the performance of our switching architecture in terms of packets average delay and buffer requirements.

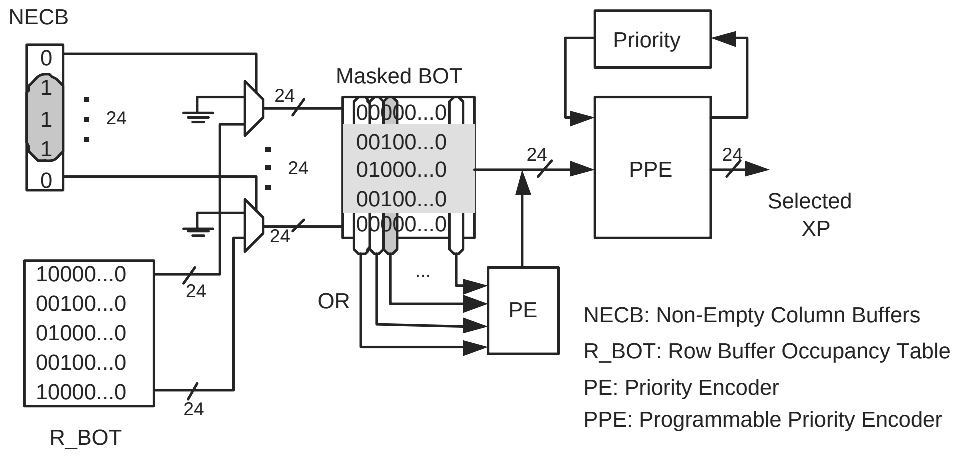


Figure 5: Output Arbiter Micro-Architecture.

Table 2: Area Results

Module	Slices	Instances	Total Slices
Input Arbiter	1197	24	28728
Output Arbiter	632	24	15168
Column BOT	28	24	672
Row BOT	28	24	672
FIFOs	19	529	10051
Total Slices			55291

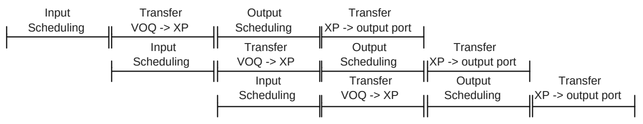


Figure 6: Packets flow.

While the performance of the IBC switching has been studied previously, the actual study is carried with respect to our specific design and target FPGA device.

4.1 Implementation Results

In this section, we present the implementation results in terms of timing and area. The design is mapped to a Xilinx Virtex4-FX device and the results are presented after place and route. The arrival rate of 64-Bytes packets at $OC-192$ line rate is one packet every 51.2 ns. The Rocket IO transceiver can be configured to de-serialize the input into a $64-bit$ wide bus at 156 MHz (64 x 156 x $(10)^6$ $\approx$ 10 x $(10)^9$). The clock cycle time is 6.4 ns, hence each packet can be transferred in 8 cycles (6.4 x 8 $=$ 51.2 ns). The input arbiter has been designed to work in the same clock frequency and has been divided into 8 cycles. Hence, while a packet is being transferred from the input card to the crossbar switch, a new queue is selected by the input arbiter, as depicted in figure 6. The critical path of the design is the Priority Encoder used to forward the selected vector to the programmable priority encoder. This module is made of a $24-to-1$ 24 bit multiplexer, checking 24 bits to decide which vector is selected.

Table 3: Percentage of Resource Allocation

Module	Instances Used	Available	Percentage
BRAMs	529	552	95.83 %
RocketIO	24	24	100 %
Slices	55291	63168	87.3 %
Pins	288	896	32.14 %

The area results of the implementation into a Virtex4-FX140 are depicted in Table 2. The allocation of the resources is shown in Table 3. Please note that the number of crossbar buffers is 23×23 (529) and not 24×24, since the transmission of cells from the same input and output indexes does not need to go through the crossbar fabric.

4.2 Simulation Results

This section presents some simulation results of the implemented algorithms for our proposed design. While the performances of MCBF, LQF-RR and OCF have been studied previously, in this section we present their performance with respect to our specific proposed architecture and under more realistic traffic patterns. The 18 $Kbit$ Block RAMs (BRAMs) of the FPGA device have been used as internal cross point buffers which meant that every cross point buffer can hold up to 36 cells (64 Bytes each). We simulated MCBF with LQF-RR and OCF-OCF using a 24×24 IBC switch. Each point in the resulting figures is obtained for $500,000$ time slots (cell time), and the statistics are gathered from the $(50,000)^{th}$ time slot. The performance evaluation is done through two non uniform traffic patterns: the diagonal traffic as defined in [4] and the unbalanced traffic as defined in [11].

As depicted in Figure 7 and 8, the average delay of the algorithms used in our design is closely comparable to the average delay of these algorithms when running on the same switch but with just one cell as internal buffer size and a speed up of two[1]. Please note that MCBF_x2 refers to the IBC switch running the MCBF scheduler, using an internal

[1]A speed up of two means that the crossbar fabric runs twice as fast as the input/output ports.

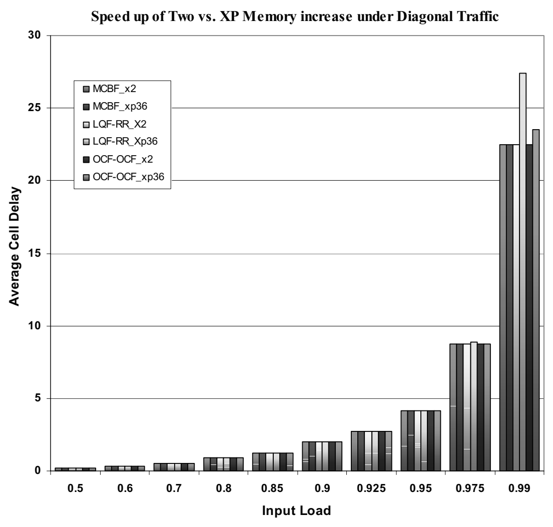


Figure 7: Average delay comparison between using a speed up = 2 and internal buffer size per cross point = 36 cells, under diagonal traffic.

crosspoint buffer size of just 1 cell and running at speed up of two, while MCBF_xp36 refers to the same system but with an internal cross point buffers of 36 cells and a speed up of just one. This result suggested that we can trade speed for internal memory. However, as mentioned before, our FPGA device contains the BRAMS that can be directly used as internal buffers. Therefore, we almost achieved a speed up of two at no extra cost.

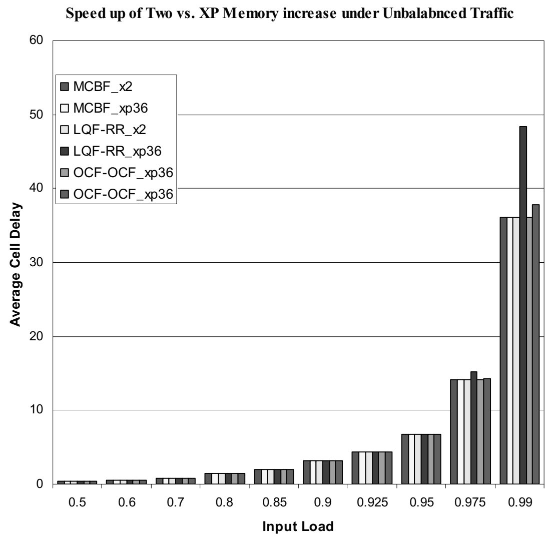


Figure 8: Average delay comparison between using a speed up = 2 and internal buffer size per cross point = 36 cells, under non-uniform unbalanced traffic, w=0.5.

One more interesting result is, that we found out that by allowing enough buffering for the internal cross points, the input line card size is no longer required to be as large. The experiments results (not shown here) suggested that our IBC switch running any of the three algorithms mentioned above and using 36 cells per cross point do not require a line card buffer of more than 16 KB.

In traditional schedulers design, where the input and output schedulers are implemented outside the crossbar fabric chip, it is hard to take full advantage of the internal buffer information. This is because, as the internal buffers size increases, extra control pins are required for flow control. Our design, however, overcomes this constraint by avoiding the requirement of extra pins irrespective of the internal buffers size. Because the schedulers are embedded within the buffered crossbar fabric, we do not need to worry about the internal crosspoint buffer size as the flow control is performed *locally* (on the same chip). This would not have been possible have the schedulers been taking place outside the crossbar fabric chip. To show the benefit of our design as compared to traditional implementations, Figure 9 shows how the MCBF delay improves dramatically as the internal buffer size increases. Please note that MCBF(1) refers to a cross point buffers size of one, MCBF(4) refers to internal buffers of size 4 cells and so on.

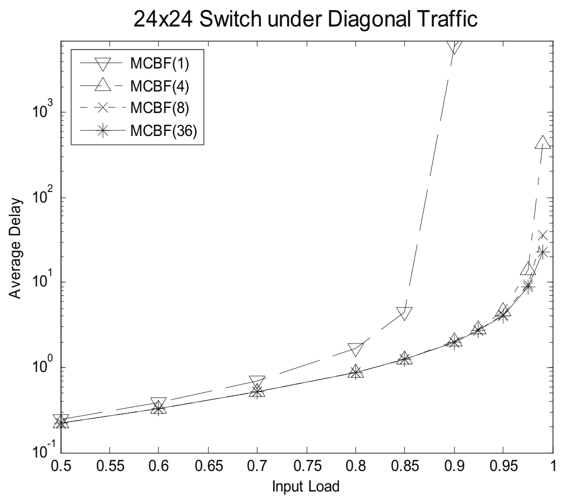


Figure 9: Delay performance of the MCBF scheduler with different internal buffer sizes under Diagonal traffic.

5. CONCLUSION

This paper proposes a new trend in designing scheduling algorithms for high-performance buffered crossbar switches. We showed the feasibility of embedding all the input and output schedulers within the crossbar chip instead of being distributed over the input and output line cards. Moving the schedulers inside the crossbar chip has the benefit of optimizing the flow control mechanism, allowing the scheduling algorithms to have faster and cheaper access to resources, with the further benefit of saving area on the input and output line cards. To show the feasibility of our design, we implemented the MCBF scheduling scheme and showed that it can fit within the crossbar chip. Additionally, we argued that our design can be easily extended to a wider range of scheduling algorithms.

6. REFERENCES

[1] Bakka, R., and Dieudonne, M. Switching Circuits for Digital Packet Switching Network. *United States Patent 4,314,367* (Feb. 1982).

[2] Chao, H. J. Next generation routers. *Proceedings of the IEEE 90*, 9 (Sept. 2002), 1518–1558.

[3] Chuang, S., Iyer, S., and McKeown, N. Practical Algorithms for Performance Guarantees in Buffered Crossbars. *IEEE INFOCOM* (March 2005).

[4] Giaccone, P., Shah, D., and Prabhakar, B. An implementable Paraller Scheduler for Input-Queued Switches. *IEEE Micro 19*, 01 (Jan./Feb. 1999).

[5] Gupta, P., and McKeown, N. Design and Implementation of a Fast Crossbar Scheduler. *IEEE Micro 19*, 01 (Jan./Feb. 1999).

[6] Javadi, T., Magill, R., and Hrabik, T. A high-Throughput Algorithm for Buffered Crossbar Switch Fabric. *IEEE ICC* (June 2001), 1581–1591.

[7] Katevenis, M., Passas, G., Simos, D., Papaefstathiou, I., and Chrysos, N. Variable packet size buffered crossbar (cicq) switches. *IEEE International Conference on Communications (ICC 2004) 02* (June 2004), 1090–1096.

[8] McKeown, N. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California at Berkeley, May 1995.

[9] Mhamdi, L., and Hamdi, M. MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches. *IEEE Communications Letters 07*, 09 (Sept. 2003), 451–453.

[10] Nabeshima, M. Performance evaluation of combined input-and crosspoint-queued switch. *IEICE Trans. on Communications B83-B*, 3 (March. 2000).

[11] Rojas-Cessa, R., Oki, E., Jing, Z., and Chao, H. J. CIXB-1: Combined Input One-Cell-Crosspoint Buffered Switch. *Proceedings of the 2001 IEEE WHPSR* (2001), 324–329.

[12] Xilinx Inc. Virtex-4 family overview. http://www.xilinx.com, March 2005.

[13] Xilinx Inc. Virtex-4 rocketio multi-gigabit transceiver. http://www.xilinx.com, March 2005.

[14] Yoshigoe, K., Jacob, A., and Christensen, K. J. The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed. *IEEE International Performance, Computing, and Communications Conference* (April 2003), 481–485.