Saturating the Transceiver Bandwidth: Switch Fabric Design on FPGAs

Zefu Dai Jianwen Zhu
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada, M5S 3G4
{zdai,jzhu}@eecg.utoronto.ca

ABSTRACT

Driven by the demand of communication systems, field programmable gate array (FPGA) devices have significantly enhanced their aggregate transceiver bandwidth, reaching terabits per second for the upcoming generation. This paper asks the question whether a single-chip switch fabric can be built that saturates the available transceiver bandwidth.

In answering this question, we propose a new switch fabric organization, called Grouped Crosspoint Queued switch, that brings significant memory efficiency over the state-of-the-art organizations. This makes it possible to build high bandwidth, high radix switches directly on FPGA that rivals ASIC performance. The proposal was validated at small scale by a 16x16 160Gps switch on the available Virtex-6 device, and simulated at a larger scale of fat-tree switching network with 5Tbps capacity.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures**]: Interconnection architectures

General Terms

Design

Keywords

Switch Fabric, Input Queued, Output Queued, Crosspoint Queued, Transceiver

1. INTRODUCTION

Recent evolution of field programmable gate array (FPGA) devices has seen a tremendous growth of transceiver speed, boasting 28 Gbps per link and Terabits per second aggregate bandwidth per device. Understandably, this is largely driven by the communication sector, reportedly FPGA's largest customer.

While the two largest FPGA vendors are drumming up the competition on delivering the highest IO bandwidth, the question remains on whether it can be fully utilized on key applications. For example, although there are announcements and reports on 100G-400G line cards, little was reported whether, and how, high radix (port count) switch fabric can be built with FPGAs that can saturate their available IO bandwidth.

The choice of switch fabric architecture is heavily influenced by the ratio of link speed and memory access speed [11]. This is because the minimum packet size is fixed and does not scale with the link speed. With higher link speed, packets arrive and leave faster. As a result, memories that implement packet buffers have to be accessed at a higher speed. To cope with the memory speed challenge, the switch fabric architecture used by application-specific integrated circuit (ASIC) based chips has evolved from the Output Queued (OQ) to Combined Input and Output Queued (CIOQ) and then to the Combined Input and Crosspoint Queued (CICQ) architecture. This is exemplified by the three generations of the IBM Prizma switch [10].

The literature on FPGA implementation of the switch fabric seems rather sparse. Early demonstrations from Actel and Xilinx [4, 24] follow the straightforward crosspoint crossbar architecture. The latest whitepaper from Altera speaks about the available device bandwidth [5], hinting the possibility of high-radix switches, but did not give design details or achieved performance of the switch itself. The NetFPGA provides an excellent platform for building low-radix switches and routers [16], but as a platform, do not discriminate specific switch architectures. The most comprehensive designs available using Virtex FPGA [25, 23] employ the CICQ architecture. But is good choice for ASICs automatically a good choice for FPGAs?

We argue that the CICQ is a bad choice for the FPGA implementation of high radix switch fabric. This is primarily due to its requirement of N^2 crosspoint buffers, where N is the port count. The SRAM resources on FPGAs will simply run out, for the large number of N permitted by modern FPGA devices. Given the known complexity of scheduling logic, the Input Queued (IQ) and CIOQ switch architectures are not favorable either.

In light of this, it becomes necessary to thoroughly investigate the switch fabric design, if the FPGA industry intends to adds it to the list of ASIC replacements, which seems to be attractive given its ubiquitous usage in carrier network, internet, data center network, and high-performance computing.

In this paper, we show that indeed it is possible to construct a high radix switch fabric using FPGAs that saturates their transceiver bandwidth: for Xilinx's Virtex-7, this promises terabits per second single-chip switching performance. In addition, in contrast to the common belief that FPGAs may suffer significant performance disadvantage, we show that the performance can rival its ASIC counterpart. More specifically, we make the following contributions:

- We propose a new switch fabric organization, called Combined Input and Grouped Crosspoint Queued architecture (GCQ), and demonstrate that it can, given the same memory resources, outperform the CICQ ar-

FPGA'12, February 22–24, 2012, Monterey, California, USA.

chitecture, the state-of-the-art switch fabric architecture;

- We show that how FPGA hard resources, instead of its logic fabric, can be best utilized to "impedance match" the GCQ architecture and give ASIC-like performance.

The rest of the paper is organized as follows. In Section 2, we review important milestones in packet switch architecture. In Section 3, we discuss the proposed switch fabric architecture in detail. In Section 4, we describe FPGA implementation issues. In Section 5, we provide evaluation result.

2. BACKGROUND

A switch fabric performs two major tasks: 1) provide datapath connections between inputs and outputs; and 2) resolve congestion. Usually, a crossbar is used to provide datapath connections between different inputs and outputs. And packet buffers are used to store data temporarily at times of congestion, i.e., when data from multiple inputs destinate to the same output simultaneously. The importance of high performance switch fabric has led to the publication of numerous switch architectures. They can be categorized into 3 basic types based on their buffer organizations:

Output Queued switch which buffers data at output ports;

Input Queued switch which buffers data at input ports;

Crosspoint Queued switch which buffers data at the crossbar;

2.1 Basic Definitions

For the following sections, we consider a switch fabric with N input and N output ports, each running at a link speed of R, e.g. 10 Gb/s. We assume packets can be segmented into fixed-length cells of size C, which are referred to as *flits*. The arrival time between flits at any input (C/R) is called a *time slot*. The *internal speedup* S of a switch is defined as the ability to remove up to S flits from any input buffer and store up to S flits to any output/crosspoint buffer at any time slot.

Table 1: Roadmap for optical fiber and SERDES speed.

	Process	nm	45	30	22
Link Char.	Link Speed	Gb/s	80	160	320
	Max Link Length	m	10		
	In Flight Data	bytes	1107	2214	4428
Optical Params	Data Wavelengths		8	16	32
	Optical Data Rate	Gb/s	10		
Electrical Params	SERDES Speed	Gb/s	10	20	32
	SERDES Channels		8	8	10

The link speed has increased rapidly over the past years from OC-3 (155Mb/s) to OC-768 (40Gb/s). However, when going beyond 10Gb/s, higher link speeds in the optical fiber are achieved by ganging multiple channels of 10Gb/s together. And the processing of such link speeds is usually done by dividing a single link to multiple parallel switches, each processing part of the data at a lower data rate, i.e. 10Gb/s. Table 1 shows the technology roadmap for optical fiber and SERDES [6]. The Dense Wavelength-Division Multiplexing technology vastly increases the number of channels available in a single fiber. But the speed of a single channel stays the same. As a consequence, low radix, high line rate (fat) switches can be implemented using parallel low-radix, low line rate (thin) switches. Therefore, in this paper, we focus on the design of more challenging high-radix switches with thin ports, rather than low radix switches with fat ports.

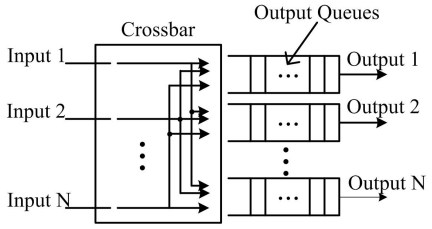

Figure 1: The Output Queued switch.

2.2 The OQ Switch

The OQ switch is the reference switch which can achieve the best possible switching performance. It has high requirements on memory access time, but low demand on packet scheduling logic. Example OQ switches include the early generation of IBM's PRS switch [9] published in 1995 and Fulcrum's FocalPoint FM4000 switch [7] published in 2009. Both switches employ the Centralized Shared Memory (CSM) architecture (which belongs to the OQ switch architecture) to achieve high throughput and low latency, as well as efficient support for multicast. The CSM architecture minimizes the amount of memory needed for congestion buffering as the memory is shared across all ports. However, the number of ports it can support is limited by the memory access time. The early generation of the IBM PRS switch supports 16 ports. Although Fulcrum's FocalPoint FM4000 switch supports 24 ports, it has exceeded the limit of memory access time. And there is a jitter of up to 70 ns when multiple output ports need to access the same memory location in parallel.

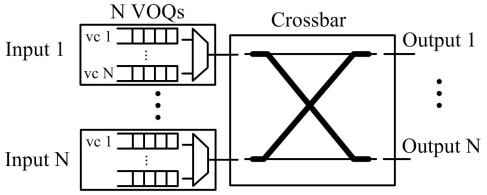

Figure 2: The Input Queued switch.

2.3 The IQ Switch

The IQ switch becomes attractive as the memory technology fails to satisfy the increasing demand on memory speed. It is advantageous that buffers in IQ switches only need to run at $2R$, compared to $2NR$ in OQ switches. However, due to the Head-of-Line (HOL) blocking problem, the throughput of the IQ switch could be limited to approximately 58% [14]. Although the Virtual Output Queue (VOQ) technique [22] can solve this blocking problem, complex scheduling algorithm is required to achieve high performance. In fact, ideal scheduling algorithms [19] are too complex to implement and practical algorithms usually take multiple stages. For example, the *Tiny Tera* IQ switch designed by McKeown et. al. at 1996 [18] uses a 3-step iSLP [17] scheduling algorithm to achieve high throughput for unicast traffic. In addition to the complex scheduling problem, the IQ switch does not support multicast well. Since all buffers are running at $2R$, a broadcast packet will take N time slots for each of its flits to pass through the switch. Otherwise, a dedicate logic has to be built for mutlticast traffic as was done in the *Tiny Tera* switch.

2.4 The CIOQ Switch

To improve the performance of the IQ switch, the Combined Input and Output Queued (CIOQ) switch has been studied and became one of the most popular switch architecture. The idea is: with an internal speedup of S, it is

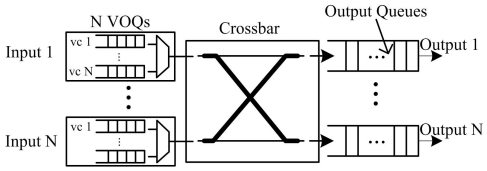

Figure 3: The Combined Input and Output Queued switch.

possible for the CIOQ switch to emulate an OQ switch with a memory speed of $(S+1)R$. Previous work has shown that with $S = 2$ and a complex centralized scheduling algorithm, the CIOQ can emulate an OQ switch [12]. However, practical implementations usually use larger internal speedup to simplify the scheduling algorithm. For example, the second generation of IBM's Prizma switch [20] is a CIOQ switch with a speedup of N. Using high internal speedup, the switch requires only a simple localized scheduler at each input port to achieve high throughput. However, despite its popularity in both academia and industry, the CIOQ switch requires either a complex centralized scheduling logic or high memory access speed.

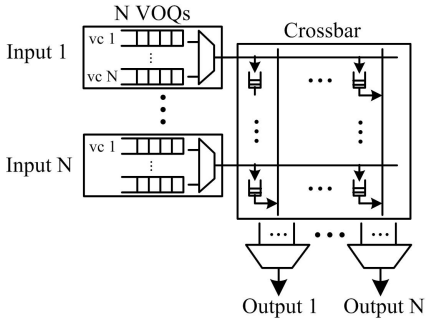

Figure 4: The Combined Input and Crosspoint Queued switch.

2.5 The CICQ Switch

The Combined Input and Crosspoint Queued (CICQ) switch was proposed to address both the complex scheduling and high memory access speed problems. The CICQ switch deploys a packet buffer in each crosspoint of its switching crossbar, so that flits from input queues are delivered to the crosspoint buffers first instead of going to output ports directly. As a result, the input and output scheduling are decoupled and no centralized scheduler is needed. Also, each crosspoint buffer only needs to run at $2R$ as it is shared by a single input and a single output port. Therefore, the CICQ switch can scale to support high port speed. Moreover, it can achieve high throughput and low latency with reasonable size crosspoint buffers [13]. These appealing features have increased the industry's interest in the CICQ switch. For example, FORTH implemented a 32×32 single chip CICQ switch [21] in 2004. And IBM's third generation Prizma switch also adopted the CICQ architecture [3] to build a 4 Tb/s single stage switch. However, the major problem of the CICQ switch lies in its requirement of N^2 crosspoint buffers, which makes it less scalable in terms of port number. Also, by distributing packet buffers across N^2 crosspoint memories, it becomes difficult to balance the workload of different buffers, resulting in low memory space efficiency in packet buffers.

2.6 The Hierarchical Crossbar Switch

In an attempt to address the port number scalability problem of the CICQ architecture and build high radix switch, Kim et. al. proposed the Hierarchical Crossbar (HC) architecture in 2005 [15]. The HC aims to reduce memory cost of the CICQ switch by partitioning the buffered crossbar into sub-switches, and implementing each sub-switch using a CIOQ architecture. It was estimated by the authors that by partitioning a 64×64 crossbar into 64 8×8 sub-switches, there is a 40% saving in memory. The HC switch can be viewed as a single stage port extension version of the CIOQ switch, as illustrated in Figure 5. Although memory saving can be achieved, complex scheduling logic is still needed in the sub-switches. In this paper, we follow a similar way of thinking as Kim's work; however, we target FPGA platform, which dictates us to address the problem of switch fabric design from a different angle. In summary, we list the resource requirements of different switch architectures including our proposed switch organization in Table 2. Details of the proposed switch organization will be explained in following sections.

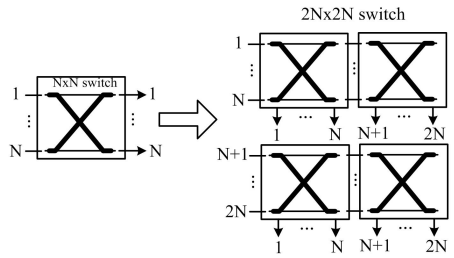

Figure 5: Single stage port extension.

Table 2: Requirement of different switch architectures.

	OQ	IQ	CIOQ	CICQ	Proposed
Input Buffer	0	N	N	N	N
Crosspoint Buffer	0	0	0	N^2	$(N/S)^2 \cdot P$
Output Buffer	N	0	N	0	0
Internal Speedup	N	1	S	1	S
Scheduling	D*	C*	C	D	D

*D = Distributed; *C = Centralized

3. MAIN IDEAS

In this section, we describe the main idea of our proposal.

3.1 Memory Is the Switch

Switching fabric often involves crossbars, well known to be wire dominated since wide multiplexers often need to be implemented. Ironically, while FPGAs are arguably made of multiplexers, they are not efficient to implement multiplexers, not to mention the long delay associated with long wires necessary to bring signals from geographical far locations to the central switch. So the conventional wisdom seems to have been that FPGAs would be inefficient, compared to the ASIC switch.

While the FPGA fabric is admittedly slow, FPGAs also integrate large amount of hardcore resources like SRAMs, which can run, in principle, at the same level of speed and power consumption as ASIC chips.

Memories have been widely used as buffers in all switches. The CICQ architecture, the most favourable today for the ASIC switches today, leverages the bandwidth of on-chip SRAMs by allocating a separate SRAM buffer at each crosspoint. Unfortunately, this does not always lead to the best utilization of SRAM bandwidth.

EXAMPLE 1. *The dual-port SRAMs on Xilinx's Virtex-6 FPGA have an access time of approximately 2.5ns. Assuming a line rate of 10Gb/s and a minimum packet size of 40B, the fastest possible packet arrival speed is 12.5ns. Therefore, one SRAM can accommodate multiple ports by way of time multiplexing. Furthermore, if multiple dual-port SRAMs can run in parallel, up to 12 input and output ports can be serviced simultaneously.*

This example illustrates that by clever organization of SRAMs both in time (by time multiplexing) and space (but using parallel memories), one can achieve a decent speedup, and as a result, one buffer can serve multiple port accesses at the same time. This leads us to the first idea that **memory can be shared**: let S be the memory speedup, a shared buffer can replace $S \times S$ crosspoint buffers.

Further examination reveals that it is not necessary to implement $S \times S$ logical crosspoint buffers. In fact, one only needs to implement S logical output buffers. This insight leads to the second idea that **memory is the switch**: while FPGA logic fabric might be slow, the FPGA SRAMs, which run at the ASIC speed, can serve as a small-scale switch: In fact, the hardwired decoding logic and sense-amplifier boosted data bus can serve the same purpose of wide multiplexers! Now the shared buffer serves the dual purposes of buffering and switching, and can be considered as a $S \times S$ OQ *subswitch*. We can therefore organize the high radix switch fabric as a two dimensional switch, in the same spirit of [15]. In the mean time, the total number of buffers has reduced to $(N/S)^2$, a S^2 reduction from the CICQ architecture.

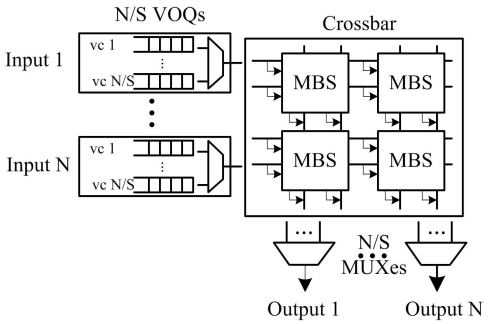

Figure 6: The organization of the proposed switch.

The third idea, which is not new, but comes as added benefit of using shared buffers, is that **memory can be borrowed**: we do not have to reserve fixed buffer size for each logical queue in the shared buffer. By using dynamic memory allocation, a "busy" output queue could occupy more memory spaces than less busy ones, and as a result, can accommodate bursty traffic or congestion better. The input queue buffers then can afford to be smaller due to less congestion burden.

We call each shared buffer a *Memory Based Switch (MBS)*, as it uses memory to implement both switching and buffering, the two major tasks of a switch.

As dipicted in Figure 6, since each shared buffer functions as a small switch, the input queues only need to maintain N/S VOQs. And the size of both input and output schedulers is reduced from $N - to - 1$ to $N/S - to - 1$, as part of the switching is done by the MBSs.

In summary, by leveraging the speedup of the SRAMs on FPGAs, we can reap the following benefits:

1. reduced number of packet buffers.
2. reduced input-queue length.
3. reduced number of VOQs in the input queues.
4. reduced complexity for both the input and output scheduling.

We name the resulting switch fabric architecture, which employs an array of memory-based subswitches, the Combined Input and Grouped Crosspoint Queued (GCQ) switch.

3.2 The Shared Buffer Design

To allow access time sharing, the memories that implement the shared buffers in the crossbar should run S times faster than the flit arrival rate. And the S input and output ports are serviced in a time-multiplexing manner, as shown in Figure 7. Now because the shared buffers in the crossbar are running at a different clock frequency, clock domain crossing logic is needed both before and after the crossbar.

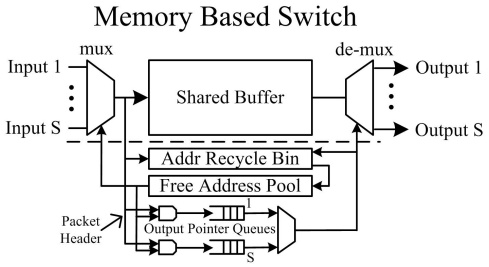

Figure 7: The structure of the shared buffer.

To achieve memory space sharing, a dynamic memory management scheme is implemented using a Free Address Pool, Output Pointer Queues and an Address Recycle Bin. Each incoming data is provided an address out of the Free Address Pool. The data is then written into the shared buffer using allocated address. At the same time, this address is pushed into destination Output Pointer Queues. The output de-multiplexer uses pointers from the Output Pointer Queues to access the shared buffer and sends data to corresponding output ports. Using the Output Pointer Queue structure, it is trivial to support multicast. The address of each multicast data is pushed into multiple Output Pointer Queues simultaneously, while only one copy of the multicast data is stored in the shared memory. The Address Recycle Bin keeps track of all references of each data in the shared buffer. When all references of a data are sent to the output, the address of that data is recycled to the Free Address Pool.

3.3 Scalability

Using the shared buffer design, the GCQ switch can, in theory, reduce the number of buffers in the crossbar by a factor of S^2. However, there are memory overhead associated with the shared buffer design. For example, it may require multiple SRAMs to provide an aggregate bandwidth of $2SR$; the Free Address Pool, Address Recycle Bin and Output Pointer Queues costs extra SRAMs to implement. Assume the implementation of each shared buffer requires P times more SRAMs compared to a single crosspoint buffer in the CICQ swith, the actual memory requirement becomes $(N/S)^2 \cdot P$. For example, if $S = 4$ and $P = 4$, there will be a 75% saving in the memory resource.

Assuming a single SRAM can provide a maximum bandwidth of B, each crosspoint buffer in the CICQ switch needs to achieve a total bandwidth of $2R$, thus requiring $2R/B$ SRAMs. Similarly, each shared buffer in the GCQ switch with a speedup of S costs $2SR/B$ SRAMS to implement. If $2R/B \geqslant 1$, each shared buffer needs S times more SRAMs compared to a single crosspoint buffer. Therefore, $P \geqslant S$. Otherwise if $2R/B \leqslant 1$, each crosspoint buffer in the CICQ switch still need at least 1 SRAM to implement, and the overhead of each Shared Buffer becomes $2RS/B$. Then the total memory saving will be $(1 - 2R/(SB))$.

EXAMPLE 2. *The 18Kb dual-port BRAMS in Xilinx's Virtex-6 have a maximum frequency of 525 MHz, and a data width of 36-bit, providing a total bandwidth of 37.8Gb/s. For a 10Gb/s line rate, the shared buffer of the GCQ switch can be implemented with an SRAM overhead of* $0.53S$. *For* $S = 8$, *the memory saving is 93%.*

4. HARDWARE IMPLEMENTATION

This section describes the implementation details of the proposed switch. We first discuss optimizations that are made in order to achieve a feasible hardware implementation. Later, we detail the hardware costs for different implementions of the proposed design. We target a 10 Gb/s link speed as it is and will likely continue to be widely used. Throughout this section, we use Xilinx's FPGA devices.

4.1 Clock and Memory Optimizations

Before hardware is implemented, the design parameter S should be determined according to the target memory technology. The 18Kb Block SRAM (BRAM) available on Xilinx FPGAs has two physical ports and a data width of 36 bits when operates in the Simple Dual Port mode. The BRAMs have an access time of around 2.5 ns on Virtex-6 (-1 speed grade) FPGA devices and 4 ns on Spartan-6 (-2 speed grade) devices. Assuming a minimum packet size of 40 bytes, the dual-port BRAMs can service up to 12 switch ports on Virtex-6 devices and 8 ports on Spartan-6 devices. However, it is impractical for FPGA designs to run at clock frequencies as high as 400 MHz. To make a feasible hardware design, we chose to have $S = 4$ and a data width of 32-byte, which requires 8 BRAMs running at 160 MHz, amounting to a total shared buffer size of 18 KB. With a 32-byte data width, the flit arrival rate is 40 MHz.

As the port logic and shared buffers in the crossbar are running at different clock frequencies, clock domain crossing logic is needed both before and after the crossbar. The input queues can be implemented as asynchronous FIFOs directly to reduce the resource overhead. Due to the large buffer size (18 KB) available in the shared buffers, the input queue buffers only need small amount of memory space to account for pipeline latencies. In order to save BRAMs, we used distributed RAMs to implement all the input queues.

4.2 Dynamic Memory Management Optimizations

To achieve a frequency of 160 MHz, the dynamic memory management logic for each shared buffer in the crossbar has to be simple enough. As shown in Figure 7, besides the data SRAMs, the dynamic memory management logic consists of the Output Pointer Queues, Free Address Pool and Address Recycle Bin. All data SRAMs operate in Simple Dual Port mode, and require straight forward read and write control logic. In current design, we assume First-Come-First-Serve service model, so that the Output Pointer Queues can be implemented as standard FIFOs. The Free Address Pool costs a single BRAM to implement multiple free address queues, and each input port is serviced in a time multiplexing manner.

The Address Recycle Bin however is not as straight forward especially when multicast support is required. For multicast packets, only one copy of the data is stored in the shared buffer, but could be accessed multiple times by different output ports. Since different outputs are not synchronous with each other, these multiple accesses could happen in different time slots. As a result, a counter is needed for each data item to maintain reference status and do proper address recycling.

A naive solution uses a bit vector to represent the destinations of each memory location in the shared buffers, with each bit corresponding to a specific output port. For every data written to shared buffers, its associated destination vector is updated with '1's, whereas each departing flit clears a corresponding bit. For each departing flit, the Address Recycle Bin checks if its destination vector becomes zero, and determines if the address should be recycled into the Free Address Pool. But the destination vectors are not easy to implement because they are not small and have to be accessed in parallel.

EXAMPLE 3. *In a 4×4 shared buffer, each destination vector consists of 4 bits. The 18 KB shared buffer can store up to 576 flits of 32-byte data and requires 576 destination vectors, which is 2304 bits in total. In each time slot, a maximum of 4 flits will leave the shared buffer. If these 4 flits all read from the same memory location, the corresponding destination vector has to be updated 4 times continuously.*

Implementing the destination vectors using LUTs will increase the area of core logic greatly and make it difficult to meet timing. Although BRAM can be used to implement large bit vector arrays, it has read and write latency of at least 1 cycle, therefore do not support continuous updating. This problem can be solved by leveraging the byte-enable feature of the BRAM. Instead of representing each destination port with a single bit, one byte is used to denote a single destination port. Then the bit vector becomes a byte vector. The continuous updates can be done with byte-enable writes. In order for destination vectors to be checked after each update, the BRAM has to be configured to work in the "WRITE-FIRST" mode, so that each write to the BRAM will result in the updated vector appearing in the output port after certain latency. As a result, the Address Recycle Bin can be implemented with a True-Dual-Port BRAM plus a check logic in the output.

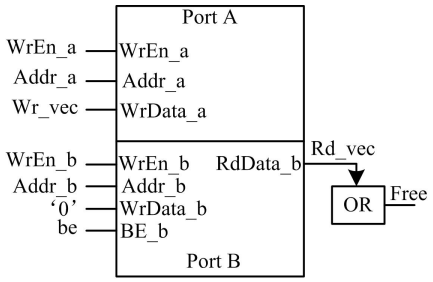

Figure 8: The Address Recycle Bin.

Figure 8 shows the implementation of destination vectors using a True-Dual-Port BRAM. For each incoming data written into the shared buffer, its corresponding destination vector is updated through Port A of the BRAM. Each departing flit will update a single byte in its destination vector using Port B with valid byte-enable signals. The updated vector will then appear in the output of Port B after a read latency. Figure 9 depicts the timing diagram of a continuous update operation. A stream of 4 updates is performed to the same address in the BRAM using different byte-enable signals. The updated data is streamed out after a latency of 1 cycle. The check logic at the output port examines the output data and generates a 'Free' signal when it finds that the output data is equal to zero.

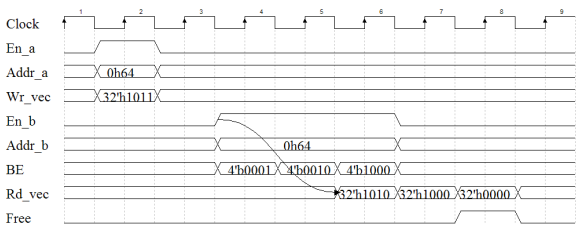

Figure 9: Timing diagram of a WRITE-FIRST SRAM.

Although feasible, the above method wastes a lot of memory space because one byte is used to represent the information of a single bit. For Virtex-6 FPGAs, this problem can be solved by exploiting the BRAM feature of independent read and write port width. For example, we can configure

Port B of the BRAM in Figure 8 to have a write width of 1 bit and a read width of 4 bits. As a result, the byte-enable signals are no longer needed and no memory area is wasted.

By examining the implementation details of all components, we show that the dynamic memory management logic of the shared buffer can be implemented with SRAMs and built-in FIFOs.

4.3 Wire Optimizations

For large switch designs, the internal wire routing is a big challenge. Considering a 16×16 GCQ switch with $S = 4$, and a data width of 32 bytes, each input port requires a 256-bit bus, which needs to be broadcast to 4 shared buffers in the same row of the crossbar. The entire switch requires a 4096-bit broadcasting bus. Given that the transceiver I/Os are typically distributed across different I/O banks, those broadcasting buses may need to travel a long distance and consume many routing resources. To alleviate this problem, we leverage the signal serialization method. As illustrated in Figure 10, the 4 input data buses are serialized into a single 256-bit bus using a 4-to-1 multiplexer, and then connected to 4 shared buffers. The serialized data bus has to run 4 times faster than the input data buses, at 160 MHz.

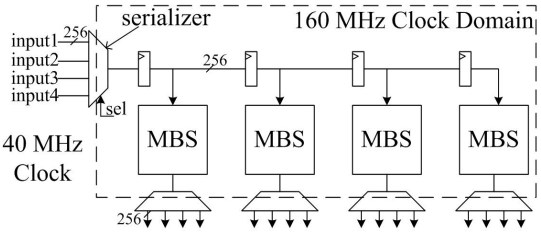

Figure 10: Wire optimization.

Signal serialization can reduce the width of the broadcasting bus, but it does not help with the long distance wiring problem. To cut the long wires down into shorter wires, registers have to be inserted. And the broadcasting bus becomes a multi-stage pipeline. The shared buffers connected to different stages of the pipeline therefore need to adjust their data arrival latency accordingly.

4.4 Hardware Cost

We chose 2 different FPGA devices to evaluate the hardware cost of the GCQ switch: Xilinx Virtex-6 XC6VLX240T-1 and Spartan-6 XC6SLX150T-3. The Virtex-6-240T device has a transceiver bandwidth of 158.4 Gb/s and 832 18Kb BRAMs; the Spartan-6-150T device has a transceiver bandwidth of 25.6 Gb/s and 268 18Kb BRAMs. Implementation and analysis was performed using version 13.1 of the Xilinx ISE Design Suite.

After applying the above described optimizations, the tool successfully routed a 16×16 switch for the Virtex-6-240T device and a 9×9 switch, with $S = 3$, for the Spartan-6-150T device. For our current designs, we assume the logic that transfers data into and out of the FGPA device through transceiver I/Os already exists. And we focus on the on-chip circuit implementation of the switch design. The resource utilization is listed in Table 3.

The 16×16 switch implemented on the Virtex-6 uses 224 18Kb BRAMs in total. As a comparison, a 16×16 CICQ switch has 256 crosspoint buffers. For each crosspoint buffer to run at 160 MHz, it requires at least 2 18Kb BRAMs to provide the data width. This results in a total of 512 BRAMs. Therefore, the proposed switch architecture is able to save 288 BRAMs for $S = 4$, which is 56% of the original requirement.

In the Spartan-6 case, the 9×9 switch costs 95 18Kb BRAMs in total. Compared to a 9×9 CICQ switch, which requires at least 162 18Kb BRAMs to run at 120MHz, the proposed architecture achieves a saving of 67 BRAMs (41%) for $S = 3$. In Spartan-6, each 18Kb BRAM in Spartan-6 FPGAs can be used as two independent 9Kb BRAMs, enabling more efficient implementation of the control logic in the core switch. For example, the Output Pointer Queues and Free Address Pool favors smaller BRAMs with narrower data width, because they only deal with small pointer size. Therefore, the dynamic memory management logic can be packed into fewer number of 18Kb BRAMs.

Table 3: Resource Utilization.

	Virtex6-240T	Spartan6-150T
N	16	9
S	4	3
Data Width	256 bits	256 bits
Core Frequency	160 MHz	120 MHz
Latency	250 ns	250 ns
Registers	36945(12%)	27028(14%)
LUTs	49537(32%)	37285(40%)
BRAMs	224(27%)	95(36%)
BRAM Saving	288(56%)	67(41%)

Both designs use many LUTs and registers to implement the clock domain crossing logic, and the broadcasting data buses. In total, the designs cost about 32% of total LUT resource of the target Virtex-6 device and 40% of the Spartan-6 device. The cost of the clock domain crossing logic can be further reduced by employing the source synchronous clock domain crossing technique in future design. The transceiver bandwidth is successfully saturated in both devices. The efficient utilization of the hardware resource provides large room for other applications, so that they can easily integrate with the switch design.

Both implementations achieve a port-to-port latency of 250 ns, most of which is spent on the clock domain crossing logic. But, we did not consider the latency of data going into and out of the FPGA through the transceiver I/Os, which normally has a total latency of around 50ns. Compared to the ASIC design of Fulcrum's FocalPoint FM4000 switch published in 2009 [7], which has a latency of 300 ns, the proposed design performs closely.

Since the proposed design is very symmetric in physical layout and only simple control logic is involved, it is possible that the speed and quality of the Place & Route can be greatly enhanced with the help of manually BRAM placement. And a higher internal speedup can be expected. Furthermore, the dynamic memory management logic of the shared buffer in the crossbar is necesitated by the support multicast traffic. If only unicast is required, the memory overhead associated with the dynamic memory management can be eliminated. And the proposed design will achieve better resource saving as discussed in Section 3.3.

5. PERFORMANCE EVALUATION

Having validated the implementation feasibility with a 16x16 switch with 160Gps switching performance, which saturates the bandwidth of the device available to us, we now turn to evaluate the the strength and weakness of the general GCQ switch architecture relative to those in the literatures, in particular, the CICQ architecture.

5.1 Evaluation Setup

To evaluate switch architecture performance, we implemented a cycle-accurate C model of the proposed switch architecture, and integrated in to Booksim, a comprehensive interconnection network simulator [1], supplied as a companion for a classic textbook on the subject [8]. The simulator provides standard traffic generators, different network topologies and performance measurement facilities, therefore enabling the fair comparison of different architectures.

We compare proposed GCQ switch architecture with IQ,

OQ, and CICQ switch architectures. For GCQ, we provide two variants: GCQ-S4, for internal speed up $S = 4$, and GCQ-S8, for $S = 8$. Key simulation configuration settings are summarized in Table 4. Here because for a flit to travel from the input queues to the crossbar it is necessary to cross clock domains, we set aside two cycles of delay. Same is true for credit packets (for flow control). The switch radix is set as 16, same as the one demonstrated in the previous section. We also simulated small packet with size of 1 flit, and size of 16 flits. The traffic is chosen as uniform traffic using an independent and identical Bernoulli process. Finally, the simulator is warmed up with traffic load before real measurement is taken.

Table 4: Simulation settings.

Flit delay	2
Credit delay	2
N	16
Flit size	32 bytes
Packet size	1 flit, 16 flit
Traffic	uniform
Network Topology	fat tree
Routing	Nearest common ancestor

For fair comparison, we assume all switch architecture have the same total buffer space.

We study both the single switch (16x16) performance, as well as network performance. The network topology is chosen as a 3-level fat-tree, which contains 192 switches and 512 ports, resulting in a total of 5 Tb/s switching capacity.

5.2 Throughput Test

We first evaluate if the proposed switch is able to achieve ideal throughput. In this experiment, we fix the buffer space in the the crossbar, and *sweep* the different input queue sizes. We set a single-flit buffer for each crosspoint for the CICQ switch (CICQ-1flit). We set the equivalent (for equal total buffer space) of 16-flit and 64-flit shared buffer for GCQ-S4 and GCQ-S8. Test results are shown in Figure 11 and 12 for different packet sizes. The horizontal axis captures the depth of each input queue and the vertical axis captures the throughput. We are interested in the "knees" of the curve for different architectures.

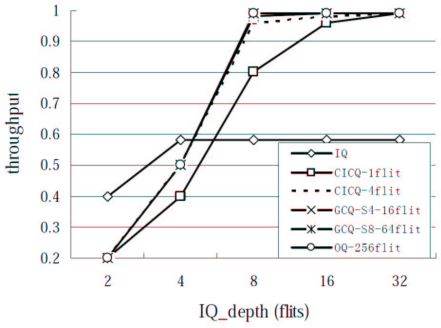

Figure 11: Throughput test with single-flit packets.

5.2.1 Small Packet

Figure 11 shows the test results using traffic consists of single-flit packets. Due to the flit traversing latency, when the input-queue-depth is smaller than 4, all switches have low throughput. The IQ switch has slightly higher throughput than others at small input-queue-depths because it only has a single stage of packet buffering inside the switch, while others buffer each packet twice inside the switches.

When input-queue-depth increases to 32, all switches except the IQ switch reach a saturation throughput close to 100%.

The CICQ switch has a much lower throughput than others at an input-queue-depth of 8. It is because the flit transferring delay between input queues and the crossbar will cause the input queue with small depth to overflow.

To see the small cross point buffer does cause problems, we also show performance of CICQ switch with 4-flit crosspoint buffers (CICQ-4flit): throughput does enhance.

The IQ switch has a saturation throughput fixed at about 58% due to the HOL blocking phenomenon.

5.2.2 Large Packet

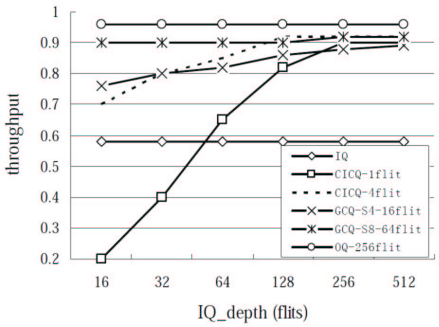

Figure 12: Throughput test with 16-flit packets.

Figure 12 shows the test results using traffic consists of 16-flit packets. With longer packets, there is higher demand on congestion buffers, as it needs much larger packet buffers to temporarily store multiple packets. The CICQ switch with 1-flit crosspoint buffer performs worst compared to other switches when the input-queue-depth is 16, because there is not enough packet buffer to resolve congestion.

The GCQ-S8 and OQ switches achieve very high throughput even with small input queue sizes because their crossbar buffers or output-queue buffers are deep enough to resolve congestion. When input-queue-depth increases to 128-flit and higher, the changes in saturation throughput of all switches become very small, indicating that the performance is now limited by the buffers in the crossbar or output queues. with an input-queue-depth of 128, the total packet buffer size of different switches is 2304 flits.

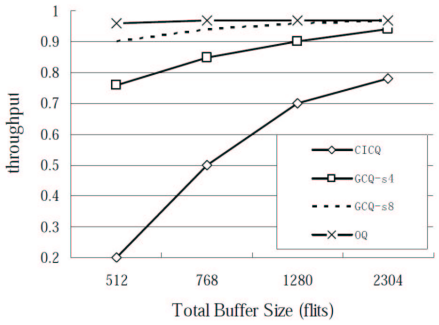

Figure 13: Throughput test with fixed IQ-depth.

5.3 Memory Efficiency

The above taught us that GCQ architecture is more "tolerant" on the input buffer space. Put it on another way, for the same input buffer space, it requires *less* packet buffers in the crossbar in order to achieve full throughput.

In this experiment, we fix the input-queue-depth as 16 flits. We then *sweep* the different cross-point buffer size and compare the throughput. For example, for a total buffer size of 2304 flits, we dedicate 256 flits of space the input queues, which left us with 2048-flit total space for others. For the CICQ switch, that means a maximum of 8-flit in

each crosspoint buffer, while the GCQ-S4 and GCQ-S8 can have a maximal buffer size of 128 and 512 flits respectively.

Using traffic of 16-flit packets, the throughput of different switches with different total buffer sizes is shown in Figure 13. The OQ switch arrives at a stable result with a total buffer size of 768 flits. The GCQ-S8 achieves the same throughput of the OQ switch at a total buffer size of 2304 flits. The GCQ-S4 performs slightly worse than the OQ switch with a total buffer size of 2304 flits. The CICQ switch performs much worse due to its N^2 crosspoint buffer requirement. Because the total buffer size will increase fast with a small increase in each of its crosspoint buffer, there is not enough buffer depth to resolve congestion when the input-queue-depth is also small.

It is interesting to note that to achieve 80% throughput, the CICQ requires 2304 flit space, whereas the GCQ-S4 requires only 512 flit space.

5.4 Latency Test

We are also interested in latency result of different switch architectures under different *traffic loads*. We chose to compare the CICQ switch with 4-flit crosspoint buffer, the OQ switch with 1024 flits of total output-queue buffers as well as the GCQ switch with $S = 4$ and different sizes of shared buffers from 8 flits to 64 flits. The input-queue-depth of different switches are set to 16 flits. The IQ switch with 80 flits in each input queue is also tested. Test results are shown in Figure 14 and 15. The horizontal axis captures the injected traffic load and the vertical axis captures the average packet latency.

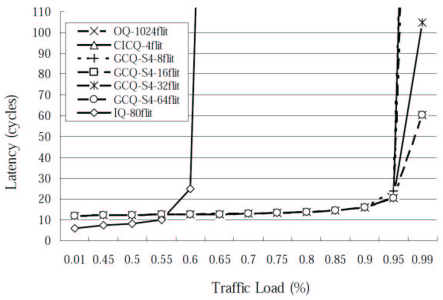

Figure 14: Latency test witch single-flit packets.

Figure 14 shows the test results using traffic consists of single-flit packets. The average latency of different switches is quite stable for a traffic load lower than 95%. For traffic loads higher than 95%, the GCQ-S4-64flit switch achieves the same performance as the OQ switch. The GCQ-S4-8flit switch performs closely to the CICQ switch. However, the total buffer size of the tested CICQ switch is more than 3 times larger than that of the GCQ switch with an 8-flit size of shared buffers.

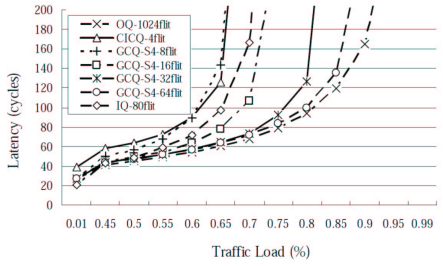

Figure 15: Latency test witch 16-flit packets.

Figure 15 shows the test results using traffic consists of 16-flit packets. Again, the GCQ-S4-8flit switch performs closely to the CICQ switch. And the GCQ-S4-64-flit switch performs closely to the OQ switch. These tests demonstrates that, with a small internal speedup of 4, the GCQ switch is able to approach the ideal OQ switch performance. And the GCQ switch is much more memory efficient in terms of performance than the CICQ switch.

Looking from another angle, with the same total buffer space, the "knees" of the GCQ curve is much more later than that of the CICQ curve. In other words, the GCQ will sustain low latency at much higher traffic load than the CICQ.

5.5 Network Performance

For scalability test, we constructed a 3-level fat-tree using different switches as the base element. As is shown in Figure 16, the fat-tree has 192 switches and 512 switch ports. Assume a 10 Gb/s port speed, this fat-tree provides a total capacity of 5 Tb/s. The Nearest Common Ancestor with random output selection is used as the routing algorithm in the fat-tree. A packet will go through a maximum of 5 hops to reach any destination port in the tree. Same as previous tests, we use uniform traffic to test the performance of the fat-tree, and results are shown in Figure 17 and 18.

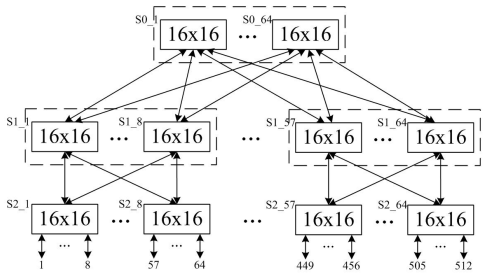

Figure 16: The Fat-tree switch network.

Figure 17 plots the test results using traffic consists of single-flit packets. The fat-trees constructed from different switch elements exhibit similar performance features as their base switches, but have higher packet latency. The GCQ-S4-32flit fat-tree achieves almost the same performance as the GCQ-S4-64flit as well as OQ fat-trees, which indicates the GCQ switch with a shared buffer size of 32 flits already has enough packet buffer for fat-tree topology in this test. The fat-tree constructed from the CICQ switch performs closely to the GCQ-S4-16flit fat-tree when the traffic load is high. The IQ fat-tree saturates at a traffic load of about 60%, which is similar to the single switch performance.

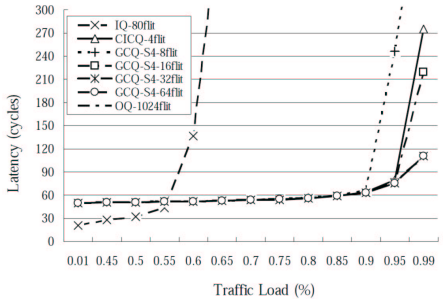

Figure 17: Fat-tree test witch single-flit packets.

Figure 17 shows the test results using traffic consists of 16-flit packets. The GCQ-S4-8flit, GCQ-S4-16flit and CICQ fat-trees quickly saturate at a traffic load of around 60%, because there are not enough buffer depth.The IQ fat-tree benefits from the long packet traffic and saturates at around 65% traffic load. The GCQ-S4-64flit fat-tree performs closely to the OQ fat-tree with a saturation throughput of approximately 85%. These results demonstrate that, the GCQ

switch with a shared buffer size of 64 flits performs closely to the ideal OQ switch with the same total packet buffer size, and is less sensitive to changes of burst length of the traffic.

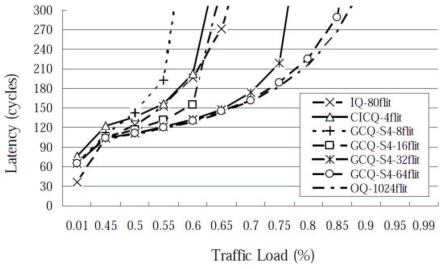

Figure 18: Fat-tree test witch 16-flit packets.

6. CONCLUSION

In this paper, we describe a new switch fabric organization and argue its merits against other organizations in general, the combined input and crosspoint queued (CICQ) switch in particular. The proposal was demonstrated by an implementation of a single-chip 16x16 switch fabric, reaching 160Gps switching capacity. This is the largest permissible for the Virtex6-240T device available to us on a ML605 development board, running at a modest frequency of 40Mhz for port processing logic, and 160Mhz for the BRAMs.

We now return back to our question raised in the beginning of the article: *is it possible to implement a single-chip switch on FPGA that saturates its transceiver bandwidth?*

Consider the largest, the latest (not yet shipped, but advertised) Xilinx Virtex-7 XC7VH870T device, which has a maximum transceiver bandwidth of about 1.4 Tbps (with 72 13.1Gps links and 16 28.05Gps links), and the maximum frequency of 600 MHz for 2820 18-Kb, 36-bit wide dual-port BRAMs [2]. For ease of calculation, consider the implementation of a high radix, 100x100 switch with 10Gps link speed, running 400Mhz for its BRAMs.

For CICQ, each cross point buffer requires 1 BRAM, therefore a total of at least 10,000 BRAMs, not including input buffers. This is far from being possible.

In contrast, with the proposed GCQ architecture with a speed up of $S = 8$, each Shared Buffer requires 6 BRAMs, therefore a total of 1014 BRAMs, about half of what is available on the device.

7. ACKNOWLEDEGEMENTS

The authors like to thank Canwen Xiao for his critiques and help in the network simulation. The authors also like to thank the support of National Sciences and Engineering Research Council of Canada, as well as China Scholarship Council for the first author.

8. REFERENCES

[1] Booksim interconnection network simulator. http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/resources/booksim.

[2] Xilinx virtex-7 data sheet. `http://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_%Data_Sheet.pdf`.

[3] F. Abel, C. Minkenberg, R. P. Luijten, M. Gusat, and I. Iliadis. A four-terabit packet switch supporting long round-trip times. 2002.

[4] Actel, Inc. Designing high-speed ATM switch fabrics by using Actel FPGAs. http://www.actel.com/documents/hispeedatm_an.pdf, 1996.

[5] Altera, Inc. Integrating 100-GbE switching solutions on 28-nms fpgas. http://www.altera.com/literature/wp/wp-01127-stxv-100gbe-switching.pdf, 2010.

[6] N. Binkert, A. Davis, N. P. Jouppi, M. McLaren, N. Muralimanohar, R. Schreiber, and J. H. Ahn. The role of optics in future high radix switch design. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 437–448, New York, NY, USA, 2011. ACM.

[7] U. Cummings, D. Daly, R. Collins, and V. Agarwal. Fulcrum's FocalPoint FM4000: A scalable, low-latency 10 gige switch for high-performance data centers. In *17th IEEE Symposium on High Performance Interconnects*, 2009.

[8] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufman, 2004.

[9] W. E. Denzel, A. P. J. Engbersen, and I. Iliadis. A flexible shared-buffer switch for ATM at Gb/s rates. *Computer Networks & IDSN Systems*, 27:611–624, January 1995.

[10] A. P. J. Engbersen. Prizma switch technology. *IBM Journal of Research and Development*, March, 2003.

[11] S. Iyer. Load balancing and parallelism for the internet. *PhD. Thesis, Standford University*, July, 2008.

[12] S. Iyer and N. McKeown. Using constraint sets to achieve delay bounds in CIOQ switches. *IEEE Communications Letters*, 7(6), June, 2003.

[13] Y. Kanizo, D. Hay, and I. Keslassy. The crosspoint-queued switch. In *IEEE International Conference on Computer Communications*, Rio de Janeiro, Brazil, 2009.

[14] M. J. Karol, M. G. Hluchyj, and S. P. Morgan. Input vs. output queuing on a space-division packet switch. *IEEE Transactions on Communication*, 35(12):1347–1356, 1987.

[15] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *32nd Annual International Symposium on Computer Architecture*, New York, NY, USA, 2005.

[16] A. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA - an open platform for gigabit-rate network switching and routing. In *IEEE Microelectronic Systems Education*, San Diego, CA, USA, June 2007.

[17] N. McKeown. Scheduling algorithms for input-queued cell switches. *PhD. Thesis, University of California at Berkeley*, 1995.

[18] N. McKeown, M. Lzzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The tiny tera: A small high-bandwidth packet switch core. In *Proceedings of Hot Interconnects IV*.

[19] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input queue switch. In *IEEE International Conference on Computer Communications*, San Francisco, CA, USA, 1996.

[20] C. Minkenberg and T. Engbersen. A combined input and output queued packet switched system based on PRIZMA switch-on-a-chip technology. *IEEE Communication Magazine*, 38:70–77, 2000.

[21] D. Simos. Design of a 32x32 variable-packet-size buffered crossbar switch chip. *MSc. Thesis, University of Crete*, July, 2004.

[22] Y. Tamir and G. Frazier. High performance multi-queue buffers for VLSI communication switches. In *15th Annual International Symposium on Computer Architecture*, HI, USA, June 1988.

[23] Xilinx, Inc. High-speed buffered crossbar switch design using Virtex-EM devices. `http://japan.xilinx.com/support/documentation/application_notes/xapp240%.pdf`, 2000.

[24] Xilinx, Inc. Building crosspoint switches with CoolRunner-II CPLDs. `http://www.xilinx.com/support/documentation/application_notes/xapp380.p%df`, 2002.

[25] K. Yoshigoe, K. Christensen, and A. Jacob. The RR/RR CICQ switch: Hardware design for 10-Gbps link speed. In *IEEE International Performance, Computing, and Communications Conference*, 2003.