THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A HIGH-SPEED ADAPTABLE PACKET SWITCH FABRIC

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERDEM EYÜP AKBABA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2013

Approval of the thesis:

THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A HIGH-SPEED ADAPTABLE PACKET SWITCH FABRIC

submitted by **ERDEM EYÜP AKBABA** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Dept., METU _____

Dr. Mustafa SANLI
ASELSAN Inc. _____

Date: 25/01/2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Erdem Eyüp AKBABA

Signature :

ABSTRACT

THE DEVELOPMENT AND HARDWARE IMPLEMENTATION OF A HIGH-SPEED ADAPTABLE PACKET SWITCH FABRIC

Akbaba, Erdem Eyüp
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ece Güran Schmidt

February 2013, 54 pages

Routers have to be fast enough to keep pace with increasing traffic data rate because of the increasing need for network bandwidth and processing. The switch fabric component of a router is a combination of hardware and software which moves the incoming packets to the outgoing ports. The access of the input ports to the switch fabric is controlled by a scheduler which affects the overall performance together with the fabric design. In this thesis we investigate two switch fabric and scheduler architectures, the well-known *i*Slip fabric scheduler and the Byte-Focal switch. We observe that these two architectures have different behaviors under different input traffic load ranges. The novel contribution of this thesis is a combined switch architecture which is composed of these two architectures that are implemented and run in parallel to selectively forward the packets with lower delay to the outputs to achieve an overall lower average delay. The design of the combined switch is carried out on FPGA and simulated. Our results show that the combined architecture has 100% throughput and a lower average delay compared to the Byte-Focal switch and the input-queued switch with *i*Slip. On the other hand, our combined switch uses more resources in FPGA than individual *i*Slip and Byte-Focal switch.

Keywords: Throughput, FPGA, crossbar switch, load-balanced switches, Byte-Focal switch, *i*Slip, input buffering.

ÖZ

YÜKSEK HIZLI UYARLANABİLİR BİR PAKET ANAHTAR ÖRGÜSÜNÜN GELİŞTİRİLMESİ VE DONANIM GERÇEKLEMESİ

Akbaba, Erdem Eyüp
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Ece Güran Schmidt

Şubat 2013, 54 sayfa

İnternet ağlarındaki bant genişliği ve veri işlenmesi ihtiyacı arttığından, yönlendiricilerin artan trafik hızına uyum sağlayabilmesi için yeterince hızlı olmaları gerekmektedir. Yönlendiriciler, yazılım ve donanımın birleşmesiyle oluşan ve gelen paketleri çıkış portlarına yönlendiren paket anahtarlarından oluşur. Giriş portlarının paket anahtarına erişimi anahtar yapısıyla birlikte genel performansı etkileyen çizelgeleyiciler tarafından kontrol edilir. Bu tezde sıkça kullanılan *i*Slip çizelgeleyicisi ve Byte-Focal anahtar yapıları araştırılmıştır. Yapılan araştırmada her iki paket anahtarının farklı trafik yük dağılımlarında farklı davranışlara sahip olduğunu gözlemlenmiştir. Bu tezdeki yeni fikir, çıkış paketlerinde genel olarak daha düşük gecikmelerin sağlanması için her iki paket anahtarının paralel olarak uygulanmasıyla oluşan ve paketleri seçerek daha düşük gecikmeleri sağlamak için çıkış portundan çıkaran birleşik paket anahtarının uygulanmasıdır. Birleşik paket anahtar yapısı FPGA üzerinde uygulanarak simüle edilmiştir. Elde ettiğimiz sonuçlara göre uyguladığımız birleşik anahtar yapısı %100 verimlidir ve Byte-Focal paket anahtarı ile *i*Slip giriş tamponlu paket anahtarına ile kıyaslandığında düşük ortalama gecikme sürelerine sahiptir. Ancak, birleşik paket anahtarı Byte-Focal paket anahtarı ve *i*Slip giriş tamponlu paket anahtarı ile kıyaslandığında daha fazla FPGA kaynağı harcamaktadır.

Anahtar Kelimeler: Verim, FPGA, matriks anahtarı, yük dengeleyici anahtarlar, Byte-Focal anahtarı, *i*Slip, giriş tamponlama.

To my family,

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor Assoc. Prof. Dr. Ece Güran Schmidt for her guidance, support, encouragement, trust, patience and valuable contributions throughout the preparation of my thesis.

I would like to acknowledge the support of ASELSAN Inc. for the realization of this thesis.

I would like to thank my colleague Mehmet Ufuk Büyükşahin for his great contributions throughout the preparation of this thesis.

The last but not the least, I express my sincerest thanks to my family and friends who have given me encourage and support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

DRR	: Deficit Round Robin
FIFO	: First-In-First-Out
FPGA	: Field Programmable Gate Array
HOL	: Head-of-Line
*i*Slip	: Iterative Round-Robin Matching with SLIP
LQF	: Longest-Queue First
MUCFA	:Most Urgent Cell First Algorithm
PIM	: Parallel Iterative Matching
QoS	: Quality of Service
VOQ	: Virtual Output Queue

CHAPTER 1

INTRODUCTION

The bandwidth demand in the Internet grows rapidly in the past few decades so routers have to work under Gbps or more operation speed. It is not reasonable to connect the distant points in the Internet, so service providers depend on each other by connecting the dots [18]. Switch fabric is the router component which forwards the incoming data to the correct output and it is a combination of hardware and software. Switch fabric is a limited hardware resource so it has to be arbitrated by a fabric scheduler. The design of fabric scheduler affects the switch fabric capacity. Therefore, the overall performance depends on the fabric performance.

Packet forwarding process is an important issue in switch fabrics because of some specific problems such as queuing delay, throughput, switch size, scalability, buffering and incoming traffic. Fabric schedulers have to forward the incoming packets to their output ports immediately. Therefore, packets which are waiting in the fabric queues should have access to fabric as soon as possible in order to minimize queuing delay.

In fabric topologies, there is a certain need for buffers because of the packets which are destined to same output port. Buffers can be found either in input ports or in output ports. They can also be found in both inputs and outputs. Fabric scheduling is an important issue in order to achieve low delay and high throughput for large capacity switch fabrics. Most of the fabrics include a centralized scheduler which increases the interconnection complexity of the switch. Some of them require a speedup larger than 1 in order to keep pace with processing speed demand.

The proposed architectures and algorithms are studied and they are compared according to their advantages and disadvantages. Some switch fabrics algorithms are DRR, PIM, *i*Slip and some architectures are the Byte-Focal, crossbar, input-buffered, shared-memory, Banyan, clos network switches. *i*Slip switch is widely used switch fabric algorithm because it has low delay values and it is highly scalable. The Byte-Focal architecture became popular recently because it has better overall performance then the *i*Slip switch. However, they have different behaviors under different loads.

Since the Byte-Focal and *i*Slip switches have different queuing delay properties, we propose a new architecture which is supposed to have better performance than the proposed architectures. The new architecture is designed to combine the two architectures in FPGA so that they work in parallel. In order to implement the combined architecture, *i*Slip and Byte-Focal switches are implemented separately and verified. Then, the combined switch architecture is implemented in FPGA and simulations are done for different traffic loads, switch size and algorithms. The combined switch has always lower delay values under different traffic loads according to simulation results. We present simulation results of the Byte-Focal, *i*Slip and combined switch in Chapter 4.

To the best of our knowledge, the combined switch approach is a novel architecture. However, there are some dynamic reconfigurable and adaptive switching approaches for QoS improvement [5]. Furthermore we carry out full FPGA implementation and performance evaluations on hardware for *i*Slip, Byte-Focal and combined switches.

In this thesis, a high-speed switch fabric is developed and hardware implemented in FPGA. The thesis starts with an introduction part where the motivation of the study and some publications on this subject are presented.

In Chapter 2, literature overview on general switch fabric architectures is introduced. Then, *i*Slip algorithm which is proposed by Nick McKeown *et al.* [1] and the Byte-Focal architecture which is proposed by Shen *et al.* [2] is discussed in detail. Our proposed combined switch architecture which is composed of the Byte-Focal switch and the input-queued switch with *i*Slip is mentioned.

Hardware implementation procedures are explained in Chapter 3. Problems faced while implementing the architectures are discussed. The combined switch architecture is also described in detail.

In Chapter 4, composed simulation platforms and setups are mentioned. Simulation results are obtained in terms of performance metrics such average delay, switch size and input traffic. Simulation results are discussed whether they are expected or not.

The thesis concludes with Chapter 5, where summary and future works are presented and discussed. In this chapter, all the work that has been done in this study will be summarized. The simulation and experimental results will be stated and evaluated. The possible future works are mentioned.

CHAPTER 2

LITERATURE OVERVIEW

2.1 Switch Fabric Architectures and Algorithms

In this section, functions, performance metrics and challenges of switch fabrics are presented. Proposed architectures and algorithms are described and some fabric scheduling terms are defined.

Router is a network element which is used to forward data packets between computer networks. Routers which are located at gateways connect data lines from different networks. The connected networks are usually two LANs or WANs or a LAN and its ISP network. Router architecture is basically composed of input and output line cards, router processor (CPU) and backplane. Routers are connected to different networks that employ different datalink technologies through input and output line cards. Today, routers are capable of routing at multi-gigabit speeds, but routing demand is still increasing.

Recent researches on router architectures mostly include specialized hardware, efficient and faster lookup algorithms and also switching fabrics. Packet switch fabric is a mechanism whose fundamental role is to forward packets from input ports to output ports. Switch fabrics and routers form the junction between connected links. Therefore, switch fabrics have the major effect on performance of the Internet.

There are lots of fabric architectures and switching algorithms proposed in the literature so far. Each switch fabric architecture and algorithm has its own advantages and disadvantages. Every application has some more important performance challenges and some less important challenges when performing packet switching process. The optimum switch fabric architecture compatible with the best effort algorithm should be selected according to the network application. There are also some adaptable switch architectures which is able to reconfigure according to different input traffic loads [5], [16].

Switch fabrics are classified into 2 categories which are Time Division and Space Division fabrics. Every packet is time division multiplexed in Time Division fabrics. They have the advantage of extending into multicast broadcast applications. Their disadvantage is the strict capacity limitation of the internal communication structure. Some examples of the Time Division fabrics are shared medium switches (shared bus architecture) and shared memory switches. There are multiple physical paths between inputs and outputs in Space Division fabrics. Packets can be forwarded through these multiple paths simultaneously so that there can be multiple input-output interconnections at the same time. They have the advantage of non-blocking, but the disadvantage of complexity.

Head-of-line blocking (HOL) is a phenomenon which limits the performance of a switch fabric in computer networking. Head-of-line blocking occurs when packets cannot be forwarded because of waiting packets at the head of the queue even if they are going to another destination.

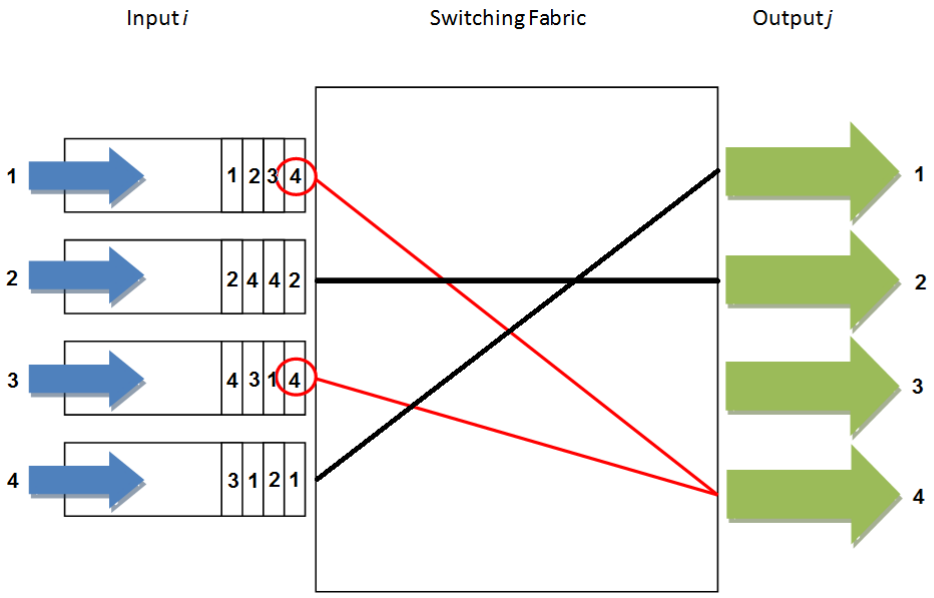

Figure 2-1 Head-of-line blocking

Head-of-line blocking is shown in

Figure 2-1. It is seen that packets at input ports 1 and 3 are waiting because of blocking packets at the head of the queues.

Virtual output queues (VOQ) are proposed in order to overcome head-of-line blocking problem. In virtual output queue mechanism, there are dedicated queues for each output port at the inputs. Virtual output queues provide a much higher throughput because of breaking the HOL blocking problem.

Fabric speedup is also an important parameter about the performance of a switch. The speedup of a switch is the ratio between the rate of traffic transferred from the inputs to an output port and the rate of the network port. It profoundly affects the control-path capabilities although the speedup is a datapath parameter. A speedup of one is a requirement for a minimal correct operation. However, as the speedup of the datapath grows, the control-path may deploy more ambitious scheduling algorithms [25].

2.2 Switch Fabrics

The need for bandwidth in network systems is increasing because data demand for recent internet applications is growing rapidly. In order to keep up with recent network applications, there has been a remarkable research about high-speed switch fabric topologies. There are lots of architectures and algorithms proposed so far. Each algorithm and architecture has its own advantages and disadvantages.

We can classify some of the proposed switch fabric architectures as shared memory switches, input buffered switches, banyan-based switches, clos-network switches and load-balanced switches. These architectures are explained in this chapter in detail.

2.2.1 Shared Memory Switches

Incoming packets are time-division multiplexed into a single data stream and sequentially written to the shared memory. In the share memory switches, outgoing packets are extracted from the shared memory and form a single output data stream. The packets are also demultiplexed into several outgoing lines. The control module checks the cell headers for output ports and extracts the memory addresses for both writing incoming packets and reading out stored packets.

2.2.2 Input Buffered Switches

FIFO queues are put in front of each input queue and they are used to store the input packets entering the switch [18], [22]. When virtual output queue structure is applied, each input has N queues for each output port. There are some algorithms applied for input buffered switches. These algorithms are namely PIM, *i*Slip, DRR and MUCFA [18]. *i*Slip algorithm is discussed in Chapter 2 in detail.

2.2.3 Banyan-Based Switches

Banyan-based switches are constructed from 2x2 crossbar switches. There is only one single path between any input-output pair. The implementation complexity is O ($N\log_2 N$) for banyan-based switches. No control mechanism is needed for routing packets from inputs to outputs; routing information is contained within each packet.

2.2.4 Clos-Network Switches

Clos-network switch is a three stage network which is defined by five parameters. r1, r2 and r3 are the number of switches in each stage, m1 is the number of inputs to a first stage switch and n3 is the number of outputs to a third stage switch. In this architecture, switches in consecutive stages are connected by exactly one edge.

2.2.5 Load-Balanced Switches

Load-balanced switch architecture is proposed to overcome throughput, delay and interconnection complexity problems [3], [4], [6], [8], [11]. Load-balanced switches use the virtual output queuing technique (VOQ) in order to solve the head-of-line (HOL) blocking problem [10].

LB-BvN switch does not need any schedulers because the connection pattern between stages is deterministic and repeated periodically [13], [19], [21]. This architecture has many advantages. First of all, the switch is highly scalable because implementation complexity is O (1). The switch has also 100% throughput and low average delay under heavy load and bursty traffic. The Byte-Focal architecture is discussed in Chapter 2 in detail.

2.2.6 Crossbar Switches

Crossbar switches are internally non-blocking and simple architectures which are composed of N^2 crosspoint switches [7], [12], [17]. Each crosspoint has two possible states which are *"cross state"* or *"bar state"*. In order to set a connection between input port i and output port j, the cross point (i, j) is set to bar state and the rest of the cross points remain in cross state.

In the literature, the architecture *"Combined Input Crosspoint Buffered Switches"* contains both input buffers and crosspoint buffers [15].

2.2.7 Fabric Schedulers

Fabric scheduling is the mechanism which packet selection is carried out. Each algorithm uses a different scheduling mechanism to exit more packets from their output ports. Buffering is also an important structure of switch fabric architectures. Virtual output queues (VOQ) which is described in detail in section 2.2 are widely used in recent fabric schedulers.

In the literature, there are lots of proposed algorithms such as *i*Slip, Byte-Focal, DRR and PIM. *i*Slip and the Byte-Focal algorithms are widely used because they are highly scalable and they have low average delay values under various traffic. They are also discussed in Chapter 2.

2.2.8 The *i*Slip Scheduling Algorithm for Input Queued Switches

The *i*Slip is a scheduling algorithm for input-queued switches which proposed by Nick McKeown in 1999 [1]. It is an input queued switch using Iterative Round-Robin with SLIP algorithm. In order to solve the contention between inputs and outputs, the *i*Slip algorithm uses round-robin schedulers. The incoming packets are initially stored in virtual output queues (VOQ). There are three steps in each iteration. It is possible to have a total of N iterations in each time slot. All inputs and outputs are unmatched at the beginning of each iteration. Inputs and outputs which are not matched at the end of iteration are available in the next iteration. The three steps for this scheme are defined as follows:

Step 1 : Request

A request is send from every unmatched input to every output for which it has a packet.

Step 2 : Grant

The output chooses the next input in a fixed, round robin schedule starting from the highest priority in the case of there are multiple requests for the same output. The output sends a grant to each input indicating whether its request was granted. The grant pointer gi is incremented (modulo N) to one location beyond the granted input if and only if the grant is accepted in step 3 of the first iteration.

Step 3 : Accept

The input chooses the next input in a fixed, round-robin schedule starting from the highest priority element. The accept pointer aj is incremented (modulo N) to one location beyond the accepted output. The accept pointers ai are only updated in the first iteration.

An example of iteration steps is shown in Figure 2-2, Figure 2-3 and Figure 2-4.

There is one packet in input port 1 which will be destined to output port 1, four packets which will be destined to output port 2. Input port 3 has two packets which will be destined to output port 2 and one packet which will be destined to output port 4. Input port 4 has three packets which will be destined to output port 4 at the same time. All input ports send requests to the outputs for which they have packets to be destined as seen in Figure 2-2.

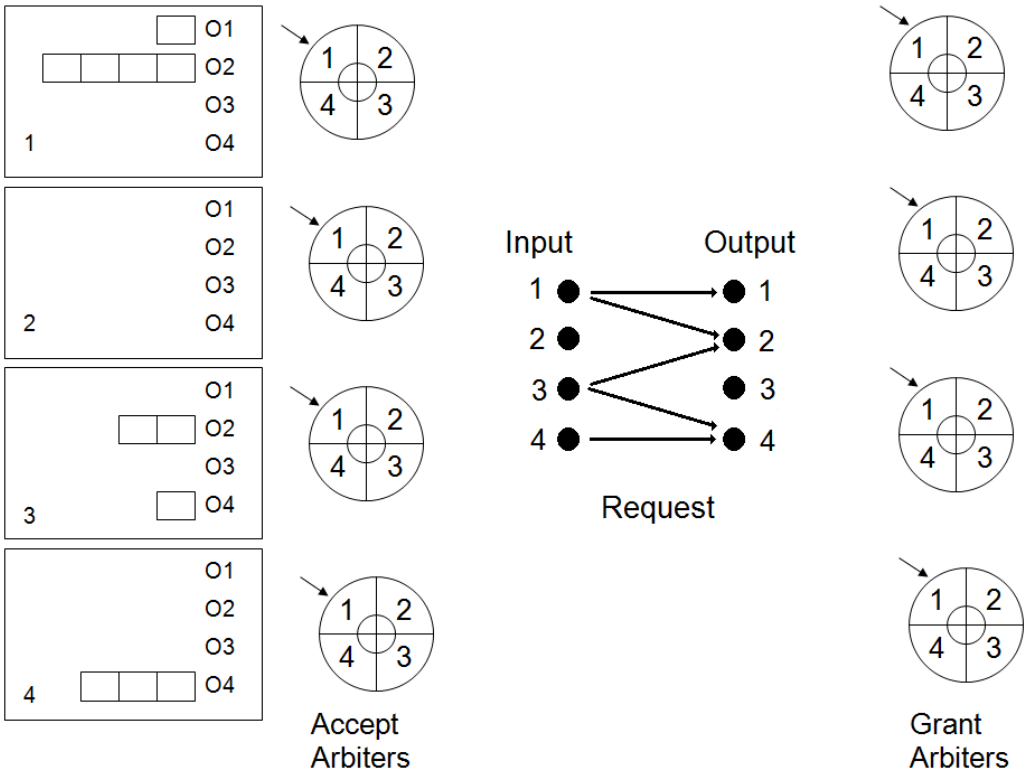

Figure 2-2 Request step (step 1) of an *i*Slip iteration

Each output port receives request signals from inputs. Then, outputs send grant signals after checking their grant pointers. If grant pointer points the input port which sends request, the output selects that input port; otherwise it grants an input port in a round-robin manner. The grant signals in our example are shown in Figure 2-3. The output 1 and output 2 chooses input port 1 and send grants to that input ports, as their grant pointers g1 and g2 point to input port 1. Grant pointer of output port 4 also points input port 1, but it receives requests from inputs 3 and 4. Therefore, it chooses input port 3 since decision of input port selection is made in a round-robin manner.

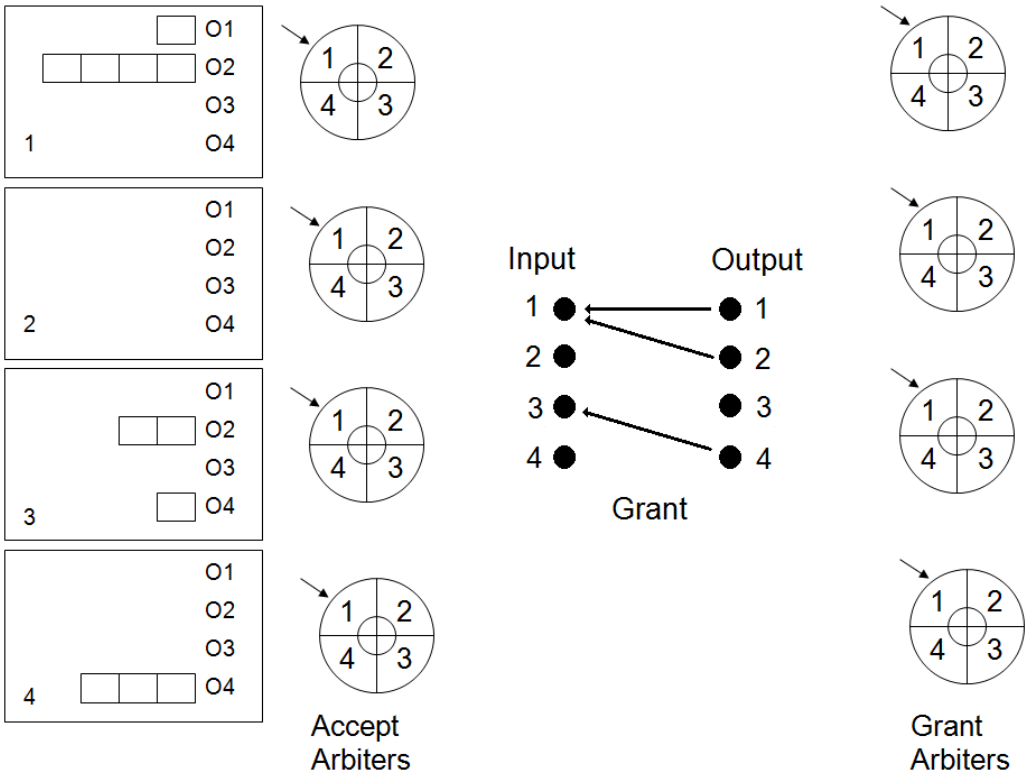

Figure 2-3 Grant step (step 2) of an *i*Slip iteration

Each input port receives grant signals from outputs after the grant step. Each input port checks that their accept pointers. If accept pointer of an input port points to the output port which sends grant signal, it accepts that output port; otherwise it chooses an output port in a round-robin manner. The accept signals in our example is shown in Figure 2-4. Input port 1 receives grants from outputs 1 and 2. It chooses output port 1 because its accept pointer points to that output port. Input port 3 receives grant from output port 4, but its accept pointer points to output 1. Therefore, it chooses output port 4 since decision of output port selection is made in a round-robin manner.

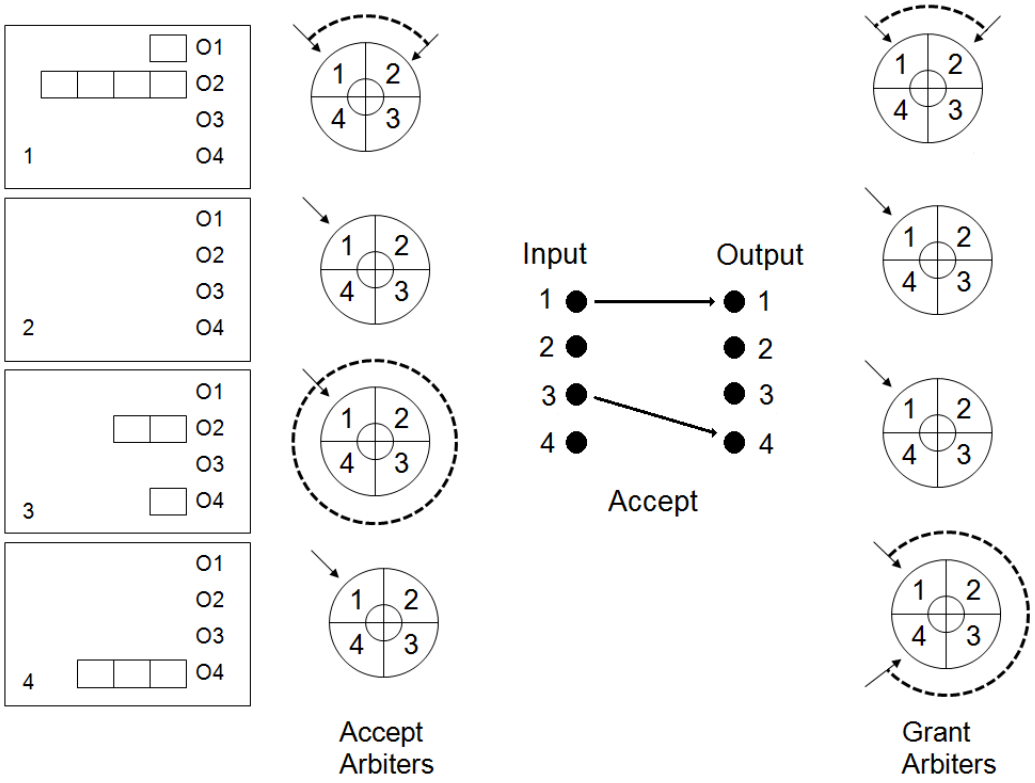

Figure 2-4 Accept step (step 3) of an *i*Slip iteration

After all steps, the packet forwarding decision is made and input ports sends their packets to the output ports which they accept. When packet forwarding process is done, all of the aj and gj pointers are updated to modulo N to one location beyond the accepted output, where N is the number of ports.

2.2.9 The Byte-Focal Switch Architecture

The load-balanced (LB) switch which is proposed by C.S. Chang *et al.* [4] consists of two stages. The first stage is load-balancing stage which converts incoming traffic into a uniform traffic. The second stage forwards packets coming from the first stage to their final destination.

The architecture of the load-balanced switch is shown in Figure 2-5. The load-balanced switch is composed of two stages and does not require any centralized scheduler. The first stage is a load-balancer which distributes the received input packets evenly into the second stage ports. It generates a uniform traffic by using a predetermined connection pattern. The second stage is a crossbar switch with virtual output queues (VOQ) and each VOQ is served at a fixed rate.

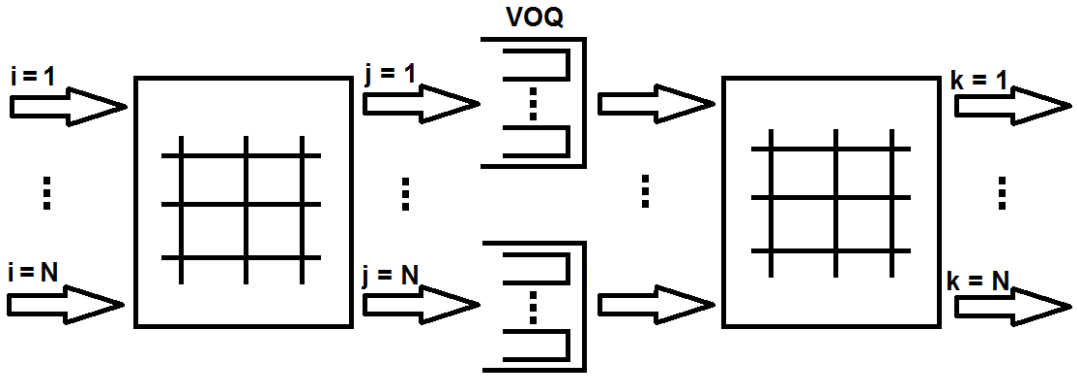

Figure 2-5 Architecture of the load-balanced Birkhoff-von Neumann switch

The load-balanced switch can guarantee 100% throughput because the second stage receives a uniform traffic due to load-balancing. However, load-balancing leads to packet missequencing at the output ports. The reason of out-of-sequence problem is that every packet can go different directions in the switch and subject to different delays.

There are many solutions in order to solve the out-of-sequence problem so far, but they are either too complex to implement or they cause a significant additional delay. In order to solve this problem, the Byte-Focal switch architecture is proposed in Shen *et al.* [2]. This architecture uses packet-by-packet scheduling in order to increase the bandwidth utilization. Therefore, the average delay performance is maximized significantly.

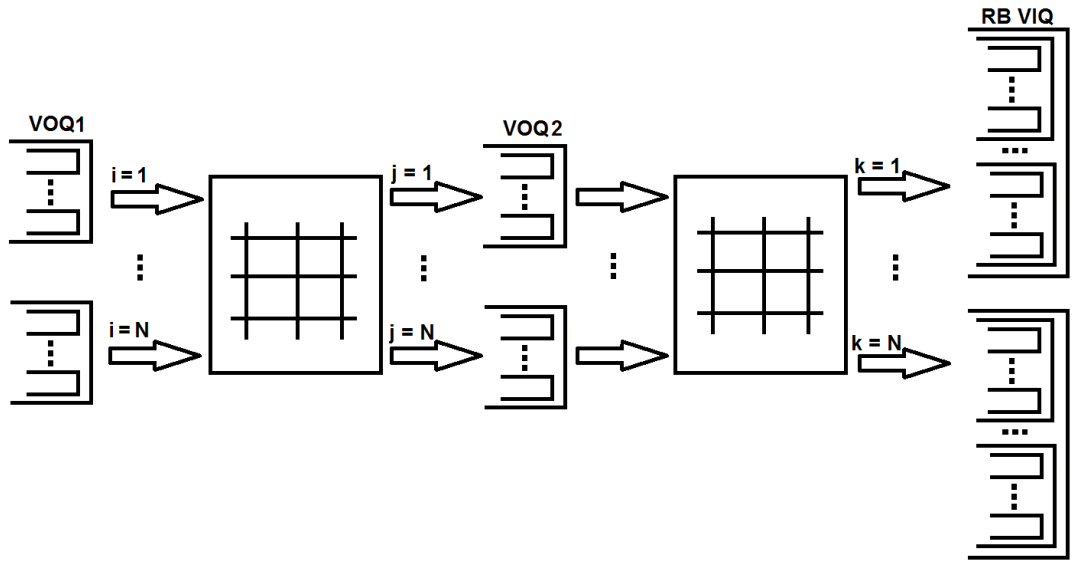

Figure 2-6 Architecture of the Byte-Focal switch

The architecture of the Byte-Focal switch is shown in Figure 2-6. The Byte-Focal switch architecture is comprised of three main stages which are namely the first stage input queue i, the middle stage queue j and the output resequencing buffer k (RB) where i, j, k = 1, 2, … N. Both switch fabrics have periodic and predetermined connection pattern.

The connection pattern (i, j) at any time slot t at the first stage is given by

j = (i + t) mod N, where i = 1, …, N and j = 1,…,N.

The connection pattern (j, k) at any time slot t at the second stage is given by

j = (k + t) mod N, where j = 1, …, N and k = 1,…,N.

If packets arriving at the input port i and destined to output port k is defined as fik, packets from flow fik are put in VOQ1 (i,k). Those packets are served in a round-robin manner such as VOQ2 (1, k), VOQ2 (2, k), …, VOQ2 (N, k) according to time slot at that instant.

Te first stage scheduling algorithm has a very important effect on the average delay performance. During the interconnection between input port i and middle stage port j, only some of the VOQ1s can be served. Therefore, each VOQ1 (i, k) has a pointer that holds the last second stage port that a packet is transferred which is called J pointer. VOQ1 (i, k) is served provided that its J pointer is pointing to the second stage input j when the first stage input i is connected to the second stage input j. After VOQ1 (i, k) is serviced, its J pointer points to the next second stage input (j + 1) mod N. Therefore, the first stage scheduling problem is:

"When input i is connected with j, each VOQ1 (i, k) whose J pointer value is equal to j sends a request to the arbiter, and the arbiter selects one of them to serve."
There are 4 algorithms for selecting a VOQ1 in order to serve the set of VOQs that can send packets to the second stage input j at time t.

1) Round-Robin:

The round-robin algorithm ensures that the arbiter makes selection in a round-robin manner. If VOQ1 (i, k) is served in a time slot, the pointer will point to the next VOQ1, i.e.. (k + 1) mod N.

2) Longest Queue First:

In the longest queue first algorithm, the arbiter makes selection to serve the VOQ1 longest queue from the set of VOQ1s that can send packets to the second stage input j at time t.

3) Fixed Threshold Scheme:

The fixed threshold scheme selects the VOQ1s that exceed a predetermined threshold (TH) length.

4) Dynamic Threshold Scheme:

The dynamic threshold scheme selects the VOQ1s that exceed a threshold (TH) value which changes dynamically with time.

The Byte-Focal switch also includes a resequencing buffer (RB) in order to solve the out-of-sequence problem at the outputs. The virtual input queue (VIQ) structure is applied in the resequencing buffers.

There are N sets of VIQs corresponding to each input port i at each output. Each VIQ set consists of N queues corresponding to a second stage input j. In Figure 2-7, it is shown for output port k = 1 that there are N sets of VIQs corresponding to each input port and there are N queues for each corresponding middle stage j in each set.

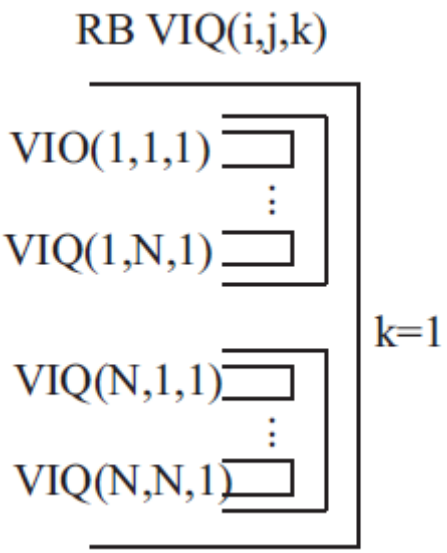

Figure 2-7 Virtual input queue structure for output 1

Packets coming from input port i and destined to output port k through middle stage port j are stored in VIQ (i, j, k). All packets in VIQ (i, j, k) are in order, because packets that are delivered in the same direction comes into the same VOQs sequentially. The advantage of using VIQs is that complexity of finding and serving packets in sequence is O (1). It is guaranteed that all packets which are exiting the output ports are in order.

2.3 Performance Metrics and Challenges

All of the proposed switch fabric architectures have their own advantages and disadvantages in terms of some specific performance metrics as mentioned previously. These metrics are namely average delay, complexity, switch size, throughput, buffers and load.

2.3.1 Average Delay

Average delay is an important performance metric for switch fabrics. Incoming packets are subject to input and output buffering delay, processing delay and also queuing delay until they exit the output ports. The most critical delay type is the queuing delay for switch fabrics. Some packets might be lost because of queuing if it is not bounded by a fixed value.

2.3.2 Complexity

Complexity is another performance metric for switch fabrics. It is also an important metric because implementation difficulty of a switch fabric is directly dependent on complexity. Byte-Focal Switch using Round-Robin, Fixed Threshold and Dynamic Threshold algorithms has a complexity of O (1), but Byte-Focal Switch using Longest Queue First (LQF) algorithm has a complexity of O (log N).

2.3.3 Switch Size

Switch size has also an important role when switch fabrics perform switching process since it has a significant effect on average delay for packets exiting from output ports. Average delay for exiting packets from output ports usually increases with switch size.

2.3.4 Throughput

Throughput is also an important performance metric for switch fabrics. A high-speed switch fabric supporting a large number of ports is expected to have a throughput of 100%. Input Queued Switch using *i*Slip algorithm and The Byte-Focal switch using Round-Robin, Longest Queue First (LQF), Fixed Threshold and Dynamic Threshold algorithms have 100% throughput.

2.3.5 Input and Output Buffers

Switch fabric architectures possess some buffers for storing the data which will be forwarded to output ports. Buffers are usually essential for packets which are waiting in queues. These buffers can exist at the inputs, outputs, or inside the fabric according to switch architecture.
Input Queued Switch using *i*Slip algorithm has N^2 Virtual Output Queues (VOQ) at input ports. There is no buffering mechanism inside or outside the fabric. On the other hand, The Byte-Focal switch has three stage buffering mechanisms. In this architecture, there are N^2 Virtual Output Queues (VOQ1) at input ports. There are also N^2 Virtual Output Queues (VOQ2) at middle stage. Finally, N^3 Virtual Input Queues (VIQ) exist at output ports.

2.3.6 Input Traffic Load

Incoming traffic load has the most significant effect on switch average delay performance. Delay experienced by forwarded packets becomes larger as input traffic load increases.

Figure 2-8 shows average queuing delays for different switching algorithms. Delay values are measured under uniform traffic for Input Queued Switch using *i*Slip algorithm and Byte-Focal Switch using Round-Robin, LQF, Dynamic Threshold and Fixed Threshold scheduling algorithms.

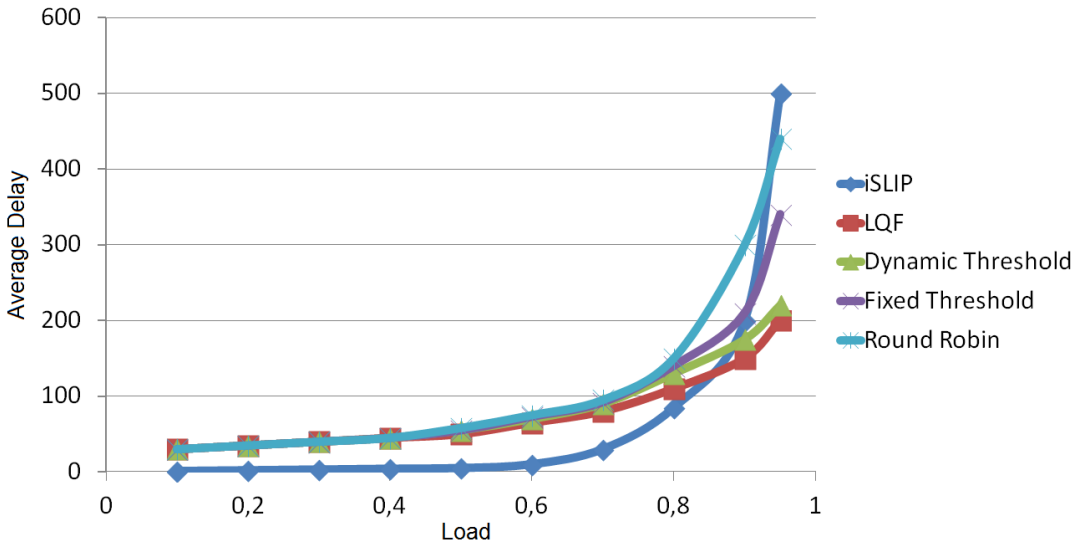

Figure 2-8 Average queuing delay for different algorithms [2]

As seen in Figure 2-8 average queuing delay for Input Queued Switch using *i*Slip algorithm is lower than Byte-Focal Switch using Round-Robin, LQF, Dynamic Threshold and Fixed Threshold scheduling algorithms under uniform traffic until load is about 0.8. However, average queuing delay for Input Queued Switch using *i*Slip algorithm becomes higher than Byte-Focal Switch using Round-Robin, Longest Queue First (LQF), Dynamic Threshold and Fixed Threshold scheduling algorithms under uniform traffic for loads above 0.8.

2.4 Comparison of the Byte-Focal and *i*Slip Algorithms

Switch fabric architecture is a combination of hardware and software which is aimed to maximize number of packets being forwarded in a unit time interval. Proposed switch architectures and algorithms have some acquisitions but they also have losses. Performance of a switch is measured according to metrics such as average delay, complexity, switch size, throughput, buffers and load which are discussed in Section 2.3 in detail.

When Input Queued Switch using *i*Slip algorithm and The Byte-Focal switch using Round-Robin, Longest Queue First (LQF), Fixed Threshold and Dynamic Threshold algorithms are compared in terms of their Average Delay vs. Input Load performance, it can be concluded that *i*Slip switch has better performance for lower input traffic. *i*Slip switch has less average delay values than The Byte-Focal switch using Round-Robin, Longest Queue First (LQF), Fixed Threshold and Dynamic Threshold algorithms under loads of 0.8 and below as seen in Figure 2-8. However, delay values for *i*Slip switch increases dramatically under input traffic loads of above 0.8. We can conclude that Input Queued Switch using *i*Slip algorithm has a better average delay performance than The Byte-Focal switch using Round-Robin, Longest Queue First (LQF), Fixed Threshold and Dynamic Threshold algorithms for lower input traffic loads and vice versa. The Byte-Focal switch has lower average delay thanks to its three-stage buffering which brings an additional implementation cost.

CHAPTER 3

IMPLEMENTATION OF THE PROPOSED COMBINED SWITCH ARCHITECTURE

The average queuing delay might be a critical issue for high-speed network applications. While working under 10 Gbps or higher network speed, even a few nanoseconds are important in order not to lose any data packets. Average queuing delay has also a significant role in real-time network applications such as VoIP. The Byte-Focal switch and Input Queued Switch using *i*Slip algorithm have different delays for different input traffic loads as we presented in the previous Chapter and shown in Figure 2-8. In this thesis we propose a combined switch architecture as shown in Figure 3-1 to have a switch fabric with lower average delay values than both algorithms.

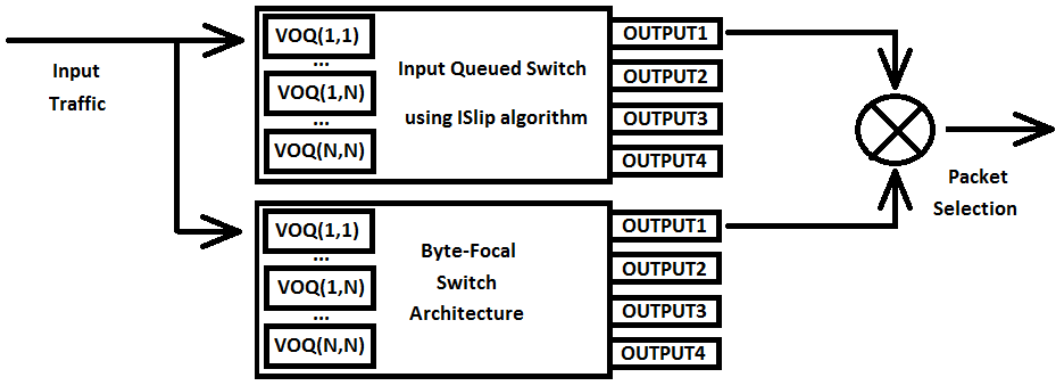

Figure 3-1 Proposed combined switch fabric architecture

The two switch fabric architectures have the same input buffering mechanism. They have both virtual output queues (VOQ) at input ports. Therefore, a combined switch which consists of the Byte-Focal switch architecture and Input Queued Switch with *i*Slip algorithm can have common input data because of having virtual output queues at input ports. It is shown in Figure 3-1 that each incoming packet is first copied and then two identical packets are inserted in the VOQs of the both switch fabric architectures. Arrows are drawn for only input and output ports 1 in the figure, but the same process is valid for all input ports.

In our proposed combined architecture; when a copy of an incoming packet appears at the output of *i*Slip or Byte-Focal fabrics *the first time*, that packet is forwarded to the output linecard to be further processed and the second copy that arrives later is discarded. Input Queued Switch with *i*Slip algorithm and the Byte-Focal switch have the same input buffering mechanisms but their output buffering is not the same, because the Byte-Focal switch has resequencing buffers (RB) at outputs.

Total delay which a packet is experienced from entering the input ports until exiting from the output ports includes both *"processing delay"* and *"queuing delay"*. Processing delay is different for the implemented Byte-Focal and *i*Slip switch. Therefore, a normalization is done on different processing delay values of the implemented architectures which is discussed in Section 3.5 in detail. Same input packets experience different queuing delays, since two switches have different architectures and different algorithms. Thus, packets which have arrived at different time intervals can exit from outputs of two switches at the same time intervals.

The packet selection mechanism in our combined switch architecture analyses the packets that are at the fabric outputs of both *i*Slip and Byte-Focal fabrics according to their arrival time and picks the earlier copy of each packet to minimize the average queuing delay and maximize the number of packets that exit the combined switch.

FPGAs are convenient for the implementation of the combined switch architecture because of their parallel operating feature with reconfigurable and adaptive processing capability. It is known that having the feature of parallel processing, very high-speed applications are designed to run on FPGAs. Therefore, implementation and verification of the combined switch architecture is possible by operating the two switch architectures in parallel and selecting the correct input packets at the outputs. If this architecture is implemented properly, there will be an important acquisition on average delay of outgoing packets.

We note that, in the combined switch architecture, there will more FPGA resource usage because input queued switch using *i*Slip algorithm and the Byte-Focal switch architectures are both implemented.

In the following of this chapter, packet switching model that we have used and data header format is expressed. Also, the Byte-Focal switch and *i*Slip switch architectures are described in detail. Finally, the combined switch architecture is represented at the end of this chapter.

3.1 Fixed-size Packet Switching Model

We have used the fixed-size packet switching model while implementing our architectures. In this scheme, the variable size IP packets are chopped into fixed size packets. This technique is widely used in the implemented routers and fabrics. Cisco 12000 series internet router architecture contains crossbar switch fabrics inside which use fixed-size packet and these cells are called "Cisco cells" [26], [27]. These cells carry parts of the IP packet as the payload and have their own headers for control. In the fixed-size packet switching model, the data payload can be selected as required. For example, cells are 64 bytes long, with an 8-byte header, a 48-byte payload and an 8-byte cyclic redundancy check (CRC) for Cisco 12000 series internet router architecture [26].

In this thesis, we have fixed size packet assumption. These packets have to be reassembled at the output port after switching process. However, this is out of our research scope. The implemented Byte-Focal and *i*Slip switch fabric architectures both designed and evaluated for fixed size packages. Since the packets are fixed size, buffer data length is arranged so that an input packet is loaded into buffer in one clock cycle. This arrangement is done for testing the switch fabrics faster and independent of the payload size. When we call "packets", the fixed-size packet headers are implied in the rest of this thesis.

3.2 Data Format

The proposed combined architecture has two different types of switch fabric architectures with different behaviors at various input traffic loads. In order to run input queued switch with *i*Slip algorithm and the Bye-Focal switch together, data types and switch timings must be adjusted so that two architectures operate coherently.

Data packet format for both switch architectures should be the because of the same virtual output queue (VOQ) buffers. Data format of input packets consist of three main fields. The first field keeps the information about the input port from which packets comes. The second field keeps the information when a packet goes into the switch. The third field shows the port to which the packet will be forwarded. Each packet has binary data which refers to the mentioned fields. Each packet header consists of a total of 24 bits.

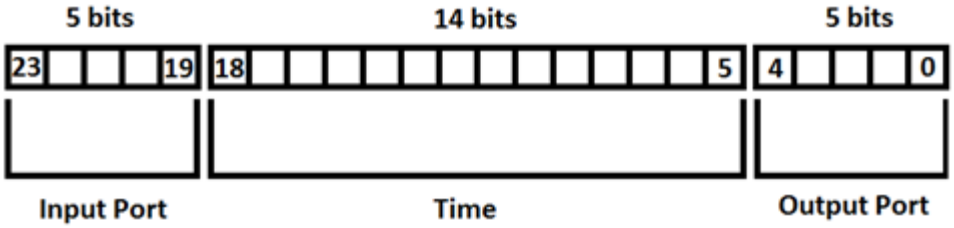

Figure 3-2 Header format for implemented switch fabrics

In Figure 3-2 it can be seen that the most significant five bits are assigned for input port, the next fourteen bits for time and the last five bits are assigned for output port information.

3.2.1 Input Port Field

We need to indicate the number of input port from which packets comes since the Byte-Focal switch architecture has three stage buffering. Resequencing buffers (RB) holds the packets according to their input port i, middle stage port j and output port k where i, j, k = 1, 2, …, N. When a packet comes to an output port, it is essential to know that packet is destined from which input port before writing it to correct resequencing buffer. Therefore, we assigned five bits of a packet header as input port data field.

3.2.2 Time Field

Combined switch architecture is intended to have low average delay values of input queued switch with *i*Slip algorithm and the Bye-Focal switch as they work together. In order to make the measurement of average delay values possible, it is required to have a time field on the packets entering the switch. Every incoming packet will carry the information about when it entered the switch. The difference between the time a packet departed from output and the time it entered in input (which is present on packet data) states total delay that packet experienced in switch. In our designed switch architecture, every packet has a 14 bit time field which means that we can generate 214 different packets in our system.

3.2.3 Output Port Field

VHDL test bench for Xilinx ISE compiler is used when doing tests of switch fabrics. The output port field of packets is needed during average delay tests and measurements. A total of five bits is assigned to indicate which output port a packet goes out.

3.3 Implementation of Input queued switch using *i*Slip algorithm

*i*Slip is one component of our combined switch fabric architecture. The architecture of input queued switch using *i*Slip algorithm is proposed by Nick McKeown in 1999 [1]. The proposed algorithm is discussed in detail in Section 2.2.8.

3.3.1 Buffers used in FPGA design

The switch has virtual output queues at the input ports as mentioned before. There are N input buffers at each of input ports. Therefore, there are total of N2 input buffers for virtual output queues.
An example of 4-port switch is shown in Figure 3-3. As seen from the figure, each VOQ set consists of 4 queues namely VOQ1, VOQ2, VOQ3 and VOQ4. Each set contains 128 packets depth of 24 bits. As an example, VOQ sets for input port 2 are shown as VOQ(2,1), VOQ(2,2), VOQ(2,3) and VOQ(2,4).

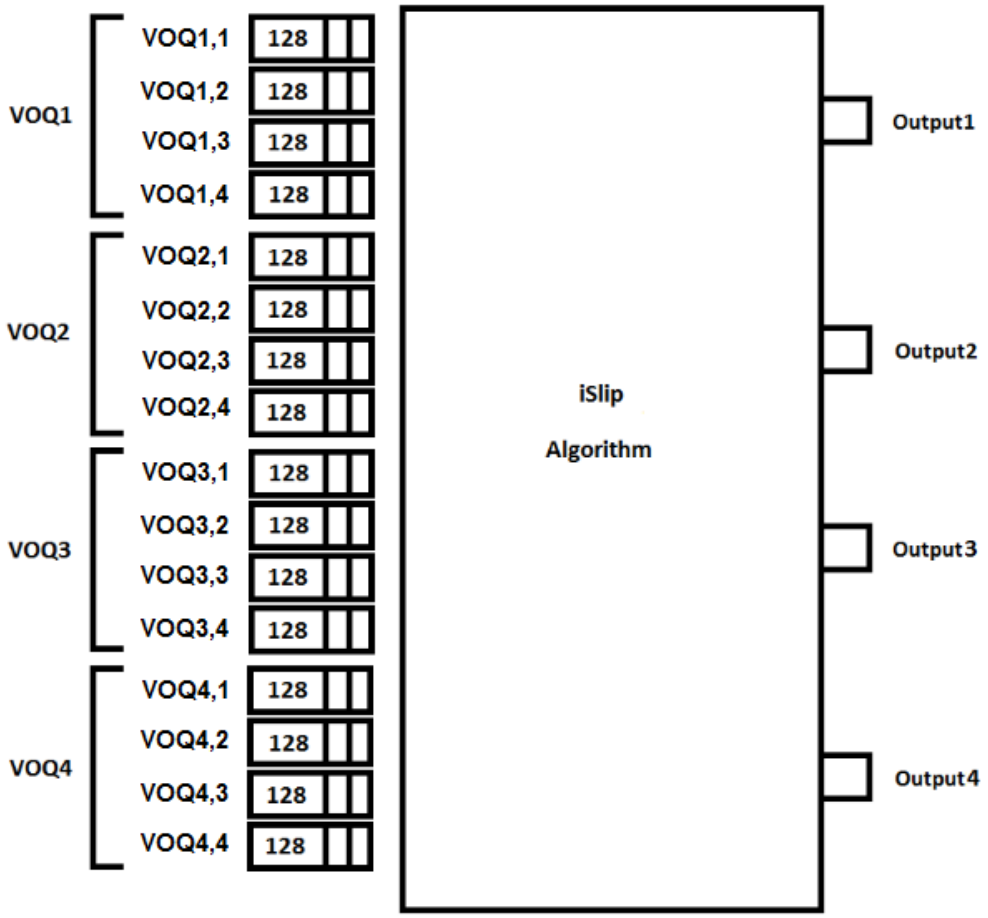

Figure 3-3 Implemented input queued switch with *i*Slip

The queues are implemented using "LogiCORE IP FIFO Generator v6.2" of ISE 13.4 project navigator tool. Block RAMs are used as memory type and read/write clock domains use common clock (clk) for this type of memory. FIFOs used in our design have signals of clock, reset, data_in, data_out, write enable, read enable, full and empty signals. Note that data_in and data_out signal are actually a bus of 24 bits. Input and output signals of each virtual output queue FIFO is shown in Figure 3-4.

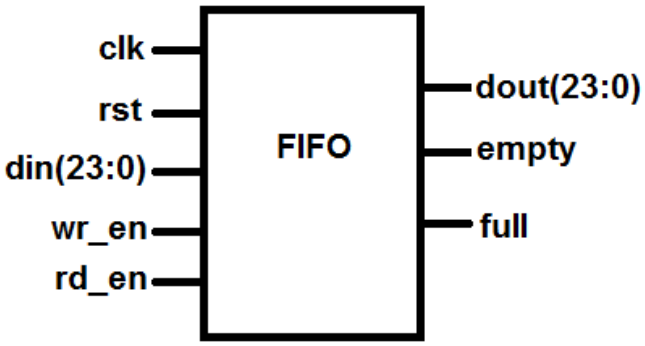

Figure 3-4 FIFOs used in VOQ implementation

3.3.2 *i*Slip Algorithm Implementation

The *i*Slip algorithm is performed by state machines in our implementation. General appearance of the generated state machines is shown in Figure 3-5. All states have active high and asynchronous reset structure which means that all state machines goes to idle state when a reset signal is asserted high.

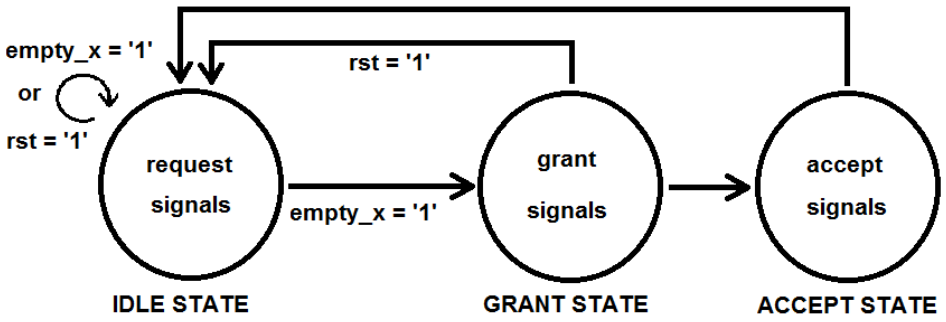

Figure 3-5 State machine structure of *i*Slip

When device power is on, the switch is initially in the idle state. The state machine also returns to the idle state when "rst" signal is asserted high.

3.3.2.1 Idle State

In the *"idle state"*, all of the configuration signals wait in their initial values. as proposed by Nick McKeown [1], request signals are initially in low state, grant pointers and accept pointers initially points to input port 1 and output port 1. In this state, *"empty"* and *"full"* signals of all queues are checked in every clock. When an input packet is written to any input queue at any instance, state machine monitors which *"empty"* signals are deasserted (i.e. which buffers has a packet), then sends request signals for input ports which have input packets.

```
625      if empty_01_03 = '0' then
626         r_i01_o03 <= '1';
627      else
628         r_i01_o03 <= '0';
629      end if;
```

Figure 3-6 Request signal assertions of *i*Slip algorithm design

Request signal definitions in FPGA design is seen in Figure 3-6. For example, if there is a packet in VOQ(3,1), the empty signal of FIFO *"empty_01_03"* will be equal to *'1'* so the request signal for that buffer *"r_01_03"* will also be asserted high in the idle state.

When all the request signal assertions are completed, the state machine will change its state to *"grant state"*.

3.3.2.2 Grant State

In the *"grant state"*, outputs check the request signals coming from inputs and send grant signals according to their grant pointers at that moment after request signals are asserted in the idle state. If a grant pointer points to the input port which sends request signal, it grants that input port, otherwise it grants the input port in a round-robin manner.

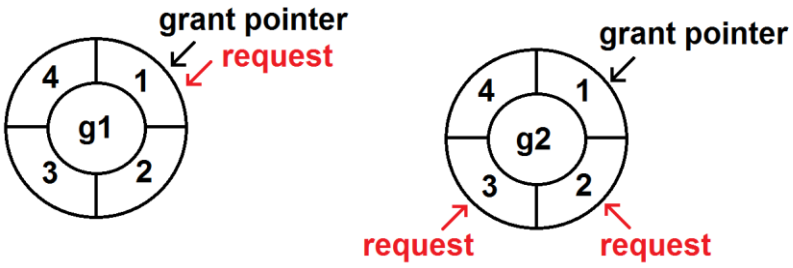

Figure 3-7 Grant pointers and request signal assertions

Figure 3-7 shows an example state of grant pointers. Grant pointer g1 for output 1 points to input port 1 and there is also a request signal *"r_01_01"* from input port 1. Therefore, output port sends a grant signal *"g_o01_i01"* which indicates the grant of input port 1. Grant pointer g2 for output 2 points to input port 1 but there is a request signal *"r_02_02"* and *"r_03_02"* from input ports 2 and 3. Output port selects an input port to grant in a round-robin manner so a grant signal *"g_o02_i02"* which indicates the grant of input port 2 is sent by output port 2.

After all the requested output ports send their grant signals to their corresponding input ports, grant state terminates and the state machine changes its state to *"accept state"*.

3.3.2.3 Accept State

In the *"accept state"*, grant signals are received by input ports. By the time grant signals are received, input ports check their accept pointers and they send accept signals to output ports which they accept. If an accept pointer points to the output port which sends grant signal, it accepts that output port, otherwise it accepts the output port in a round-robin manner.

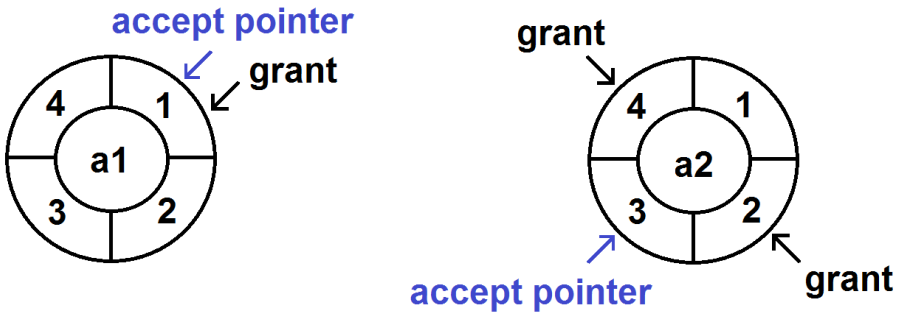

Figure 3-8 Accept pointers and grant signal assertions

Figure 3-8 shows an example state of accept pointers. Accept pointer for input 1 *"a1"* points to output port 1 and there is also a grant signal *"g_o01_i01"* from input port 1. Therefore, input port sends an accept signal *"a_i01_o01"* which indicates the accept of output port 1. Accept pointer a2 for input 2 points to output port 3 but there is an accept signal *"g_o02_i02"* and *"g_o04_i02"* from output ports 2 and 4. Input port selects an output port to accept in a round-robin manner so an accept signal *"a_i02_o04"* which indicates the acceptance of output port 4 is sent by input port 2. By the time all the granted input ports send their accept signals to their corresponding output ports, *"accept state"* is completed and the state machine changes its state to *"idle state"*. Before the state is changed, all grant pointers and accept pointers are updated to their current positions.

After the state machine returns to the idle state, all the processes mentioned in the idle state restart. If some input packets come to inputs while the state machine is trying to determine which packets will be forwarded, incoming packets have to wait in the queues until the state machine finishes its deciding process.

The time between an input packet enters the buffer of an input port and it exits the output port is 6 clock cycles. This shows the processing time of the *i*Slip switch for any packet. In this calculation, queuing delay is not included since calculation is done in an emtpy switch. Therefore, we need to have an interarrival time of 6 clocks when the switch is operating at maximum input load. In other words, the switch can forward packets to their destination outputs within 6 clock time intervals. For example, if a packet comes to an input port one clock later than the switch begins deciding process, it has to wait five more clock cycles in order to take part in deciding process.

Figure 3-9 shows packet forwarding time for input packets which come to input ports 1 and 2 at the same time when clock cycle is 10 nanoseconds. Both packets come to input ports 1 and 2 at the same time when t=0 ns. When the packets come to switch, deciding process begins to operate. The packets arrive at the outputs after six clock cycles which is equal to 60 nanoseconds.

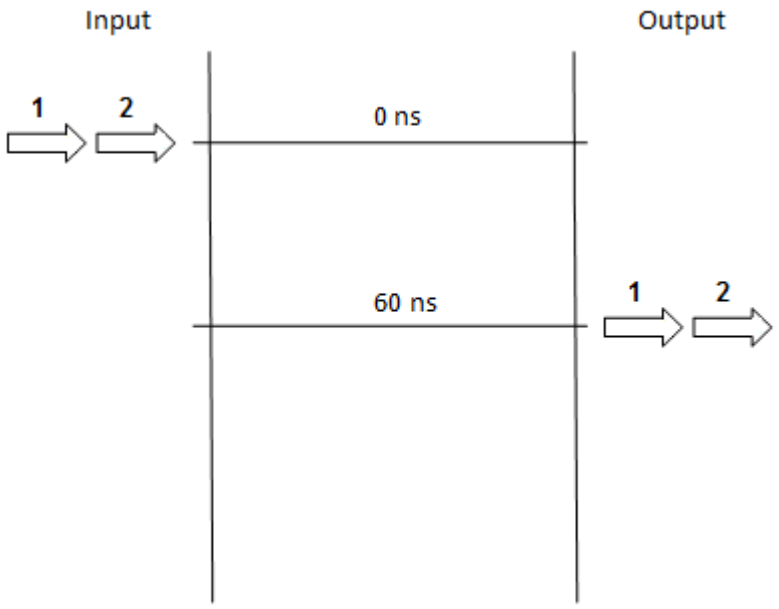

Figure 3-9 Packets times when they enter input ports at the same time

Figure 3-10 shows packet forwarding time for input packets which come to input ports 1 and 2 at different times when clock cycle is 10 nanoseconds. First packet comes to input port 1 at t = 0 ns. When the packets come to switch, deciding process begins to operate. Second packet comes to input port 2 at t=10 n i.e. 10 ns after the beginning of the deciding operation. Since the state machine is already started to operate when the second packet comes, it has to wait until the state machine process terminate. The packet 1 exists from the output 1 after six clock cycles when time is equal to 60 nanoseconds. At this time, the state machine returns to idle state and realizes that there is a waiting packet in input 2 buffer. Therefore, packet 2 arrives at the output 2 after six clock cycles when time is equal to 120 nanoseconds.

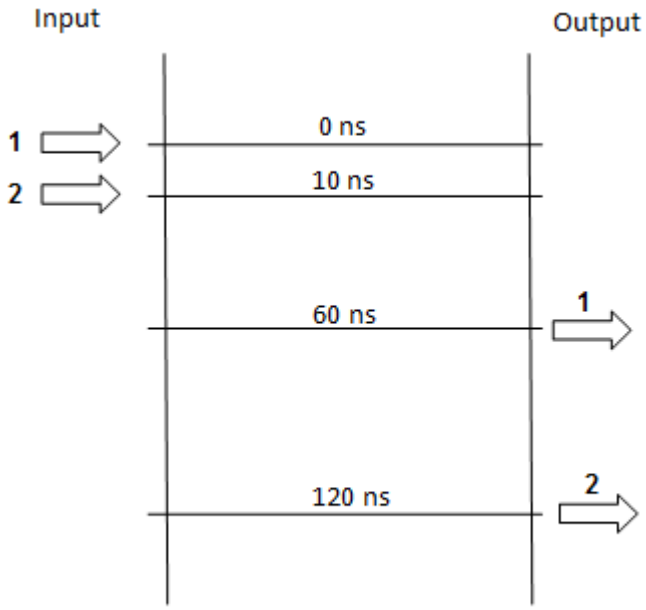

Figure 3-10 Packets times when they enter input ports at different times

As a result of this discussion, we have to send input packets with 60 nanosecond interarrival time in order to measure queuing delays correctly.

A simulation screen is seen in Figure 3-11. As seen from this figure, the departure times for outgoing packets is 60 nanoseconds when clock cycle is 10 nanoseconds.

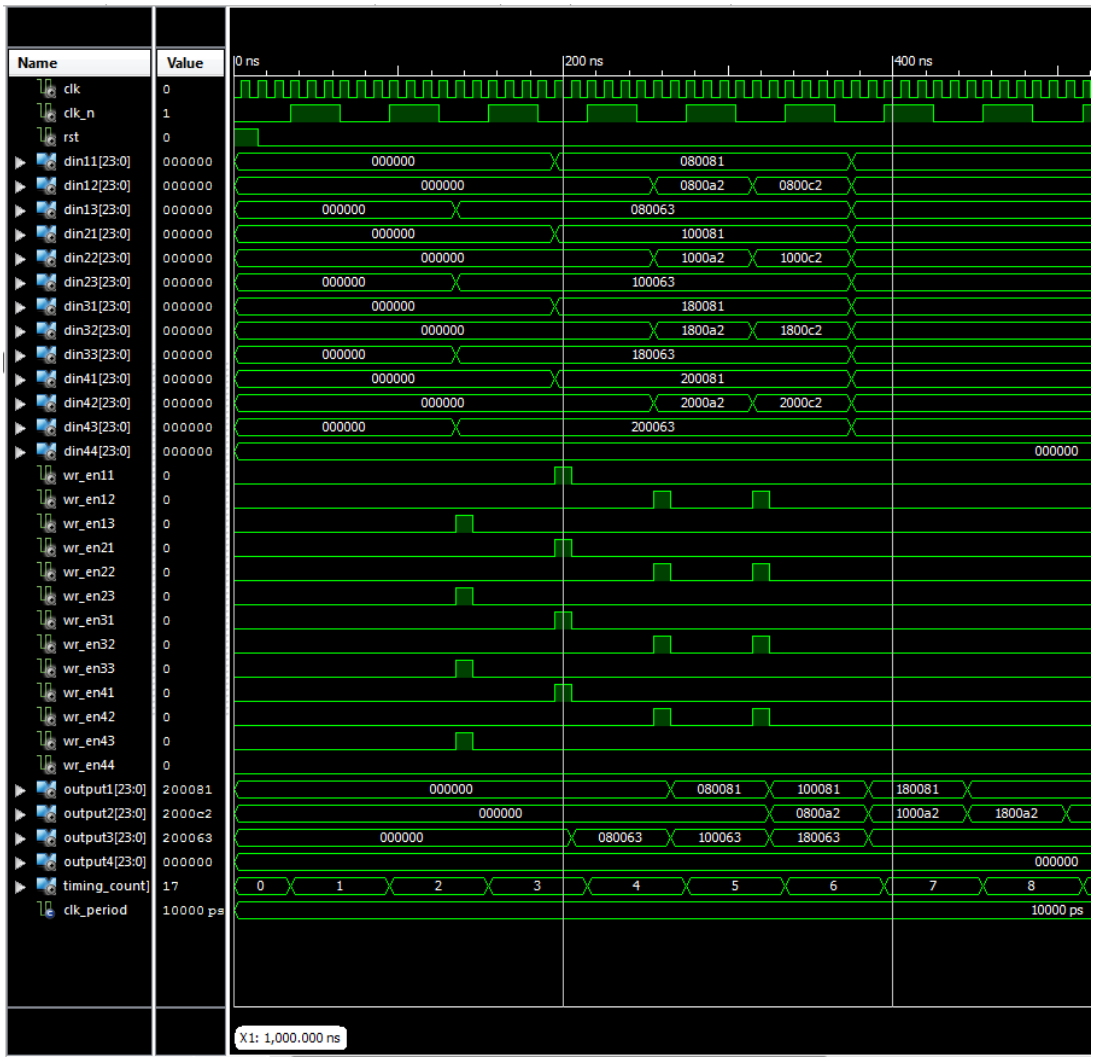

Figure 3-11 Input queued switch with *i*Slip simulation screen

Number of used resources for different number of ports is given in Table 3-1.

Table 3-1 Resources used in the input-queued switch with *i*Slip algorithm

	*i*Slip Switch	
	4-Port	8-Port
Number of Slice Registers	1307	4944
Number of Slice LUTs	1208	6185
Number of Block RAM/FIFO	8	32

3.4 Implementation of the Byte-Focal Switch Architecture

The Byte-Focal switch architecture is described in Section 2.2.3 in detail [2]. In this architecture, there are virtual output queues (VOQ) at both first (input) stage and second (middle) stage. There are also virtual input queues (VIQ) at the output ports. Virtual input queues are also called Resequencing Buffer (RB) since the out-of-order packets are resequenced at the output ports by using these buffers.

The implemented Byte-Focal switch architecture consists of three stage buffering mechanisms. The first stage and second stage buffers are standard virtual output queue (VOQ) buffers. Third stage buffers are used to put the packets in order. The implemented design in the thesis also consists of two switch fabrics which forward packets across first stage, second stage and resequencing buffers.

At the first stage, there are N sets of buffers which belong to each input port. Each buffer set contains N queues for each output port. In others words, there are a total of N2 queues in the first stage of input ports.

The same buffer architecture exists in the middle stage. A packet in the first stage buffer is forwarded to a second stage buffer according to the used algorithm in the first stage switch. A packet is written to a second stage buffer according to its output port knowledge which is included in its data. For example, if a packet in VOQ1 (2,4) is read from the first stage, it is forwarded to a second stage port according to its pointer value at that time. The pointer value depends on the used algorithm by the first stage switch and time.

The resequencing buffer structure is based on virtual input queue buffering architecture. In each output port, there are N sets of buffers and each set corresponds to an input port. In each set, there are N queues which are reserved for middle stage ports. Therefore, a total of N3 queues exist in the output ports.

The implemented architecture for a 4-port switch is shown as an example in Figure 3-12. As seen from the figure, there are four virtual output queues at input ports and each of them consists of four queues. Therefore, there are a total of 16 queues at the input ports. The middle stage has the same buffer design as the first stage, i.e., there are also a total of 16 queues at the middle stage port. Finally, there are 64 resequencing buffers at the output ports.

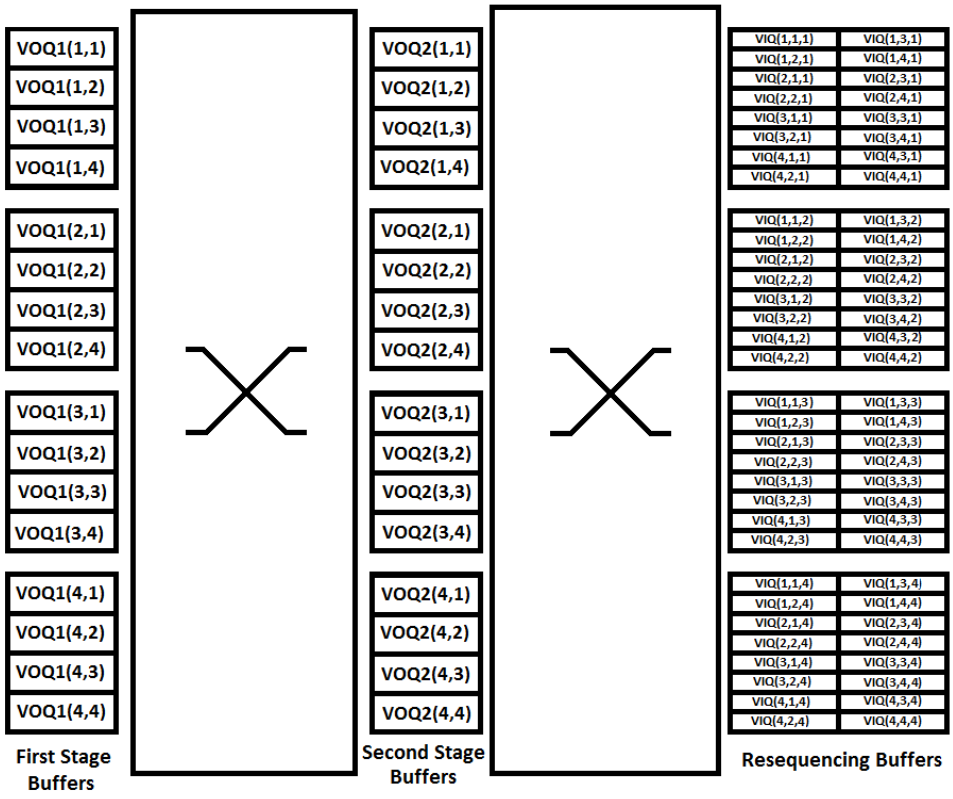

Figure 3-12 Implemented Byte-Focal architecture

3.4.1 First Stage Scheduling Algorithm

The Byte-Focal switch architecture can run with four different algorithms which are described in Section 2.2.9 in detail [2]. These algorithms are

1. Round-Robin
2. Longest Queue First
3. Fixed Threshold
4. Dynamic Threshold

In the first algorithm, the first stage switch emits packets in a round-robin manner. The arbiter selects to serve the longest queue in the second algorithm. Third and last algorithms make a selection in the queues according to a predetermined or dynamic threshold limits.

In our design, the first stage algorithm is selected as *"Round-Robin"*. The packet forwarding pointers are updated in a round-robin manner as time proceeds. Therefore, input traffic is distributed uniformly, because the first stage switch distributes incoming traffic evenly among all the second stage buffers.

The connection pattern (i, j) at any time slot t at the first stage is given by

$j = (i + t) \bmod N$, where $i = 1,\ldots,N$ and $j = 1,\ldots,N$ (3)

The connection pattern is advanced by one in every time clock. It can be concluded from the equation (3) that connection patterns are the same for every N clock cycles. Connection patterns are shown in Figure 3-13, Figure 3-14, Figure 3-15 and Figure 3-16 for a switch that has a clock period of 10 nanoseconds. Time values are shown in each figure in terms of clock cycles. Input ports i are connected to middle stage ports j where i=j at time t=0. Then the connection pattern is shifted by one for every clock cycle progress.

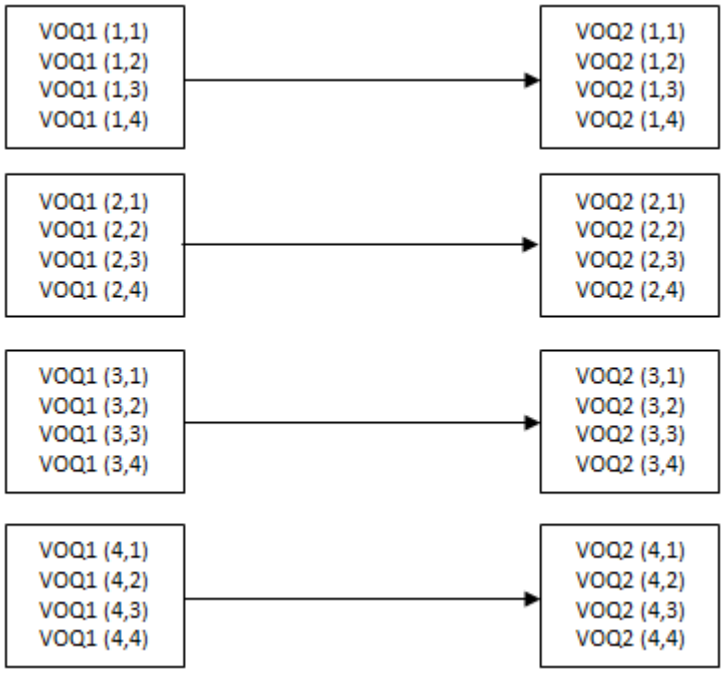

Figure 3-13 Connection pattern at time t = 0 ns

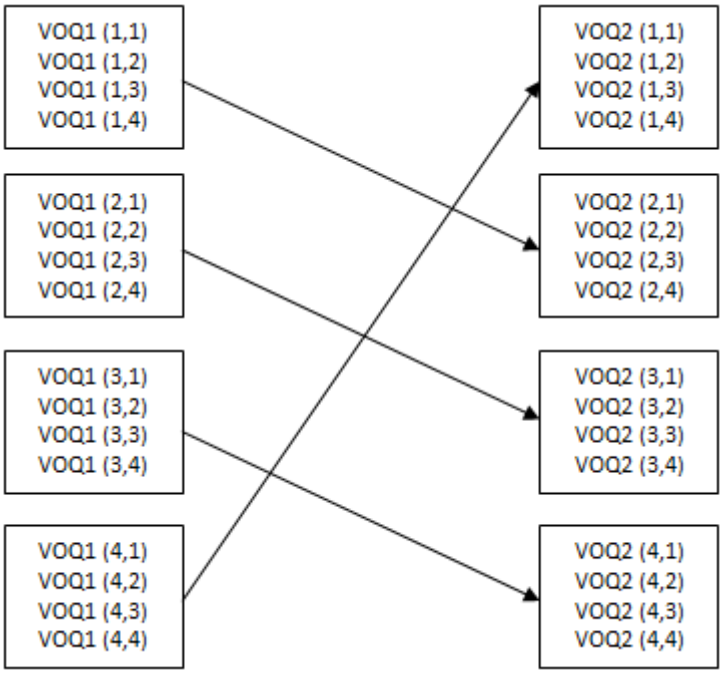

Figure 3-14 Connection pattern at time t = 10 ns

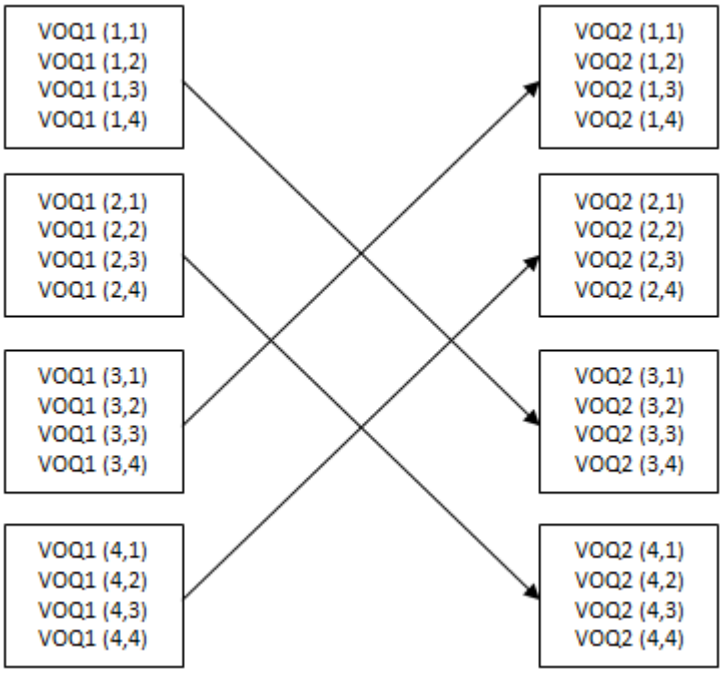

Figure 3-15 Connection pattern at time t = 20 ns

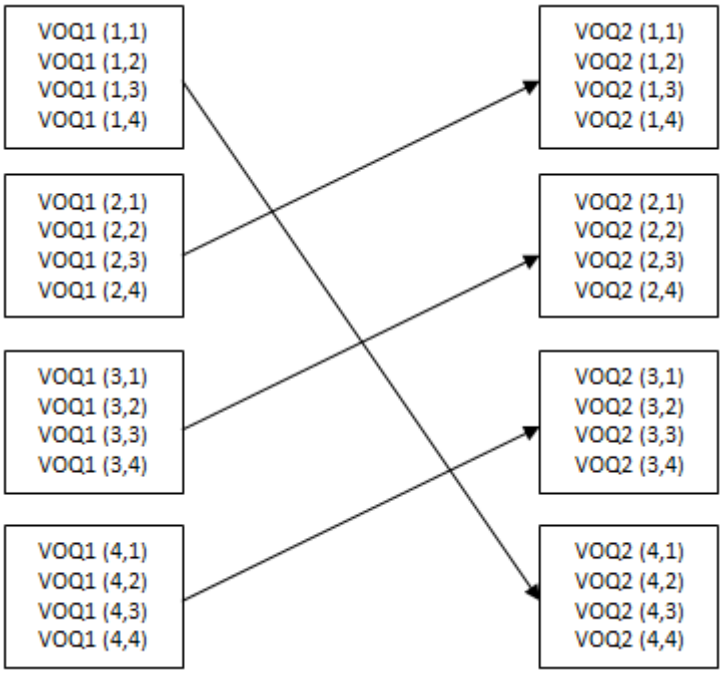

Figure 3-16 Connection pattern at time t = 30 ns

3.4.2 Second Stage Scheduling Algorithm

After the packets are read from queues at input ports, they are written to second stage virtual output queues according to their output port information which is included in its data. First stage switch uses round-robin algorithm in order to convert input traffic into uniform traffic. Therefore, an input packet is written to a middle stage buffer according to time interval.

The second stage is responsible to forward packets in middle stage buffers to resequencing buffers independent of time. Second stage is an input-queued crossbar switch, and each VOQ is served at a fixed rate [2].

Each middle stage packet is written to related queue of a resequencing buffer. Since resequencing buffers need middle stage port as well as first stage port information, we have to know the information from which input port a packet is written, we check the input port data of each packet.

Figure 3-17 shows an example connection between second stage buffers and resequencing buffers. If there is a packet in VOQ2 (2, 3), we can understand that the packet is going to exit from output port 3. Therefore, it has to be written to output port 3 buffers. We also know that the packet is in middle state port 2. However, we have to know the input port number of the packet in order to write it to correct virtual input queue. This information is included in the packet data. After reading the packet data, the state machine observes its input port data and writes the packet to correct destination. For the example, the input data information read is 4 and the packet is forwarded to VIQ (4, 2, 3).

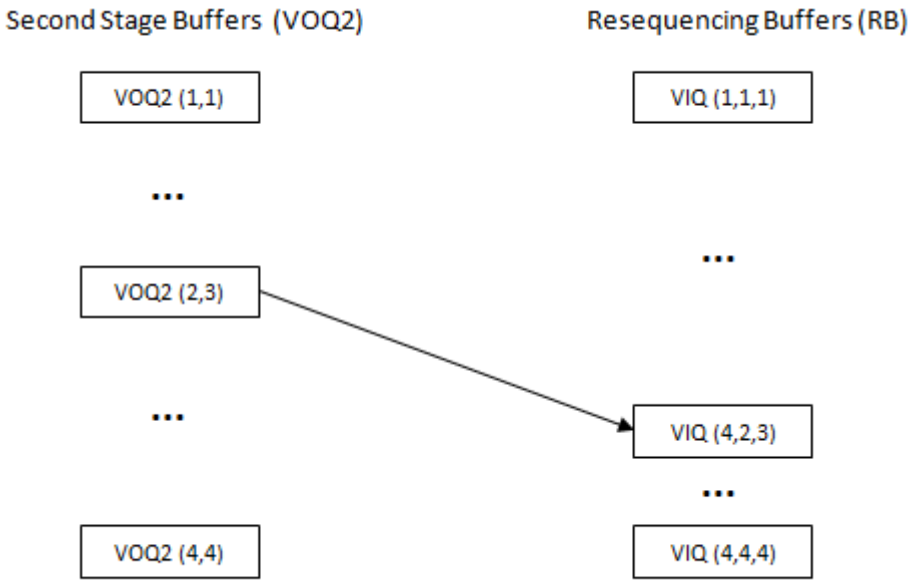

Figure 3-17 Connection between second stage VOQ and RB

3.4.3 Resequencing Buffer Design

The Byte-Focal architecture uses resequencing buffers (RB) in order to solve out-of-sequence problem [2]. There are N sets of VIQs at each output and each set corresponds to an input port i where i = 1,2,…,N. There are N queues with each queue corresponding to a second stage input j where j = 1,2,…,N. VIQ (i, j, k) organizes packets according to their input and output port as well as middle stage port information. Packets which come from input port i, and destined to output port k through middle stage port j are stored in virtual input queue VIQ (i, j, k). All the packets in VIQ (i, j, k) are in order.

Figure 3-18 shows times when two different packets enter the input ports and exits from output ports. As seen from the figure, if two packets enter different ports of the switch at the same time, they exit from correct output ports 120 ns after they enter the switch. Note that clock cycle for the switch is 10 nanoseconds.

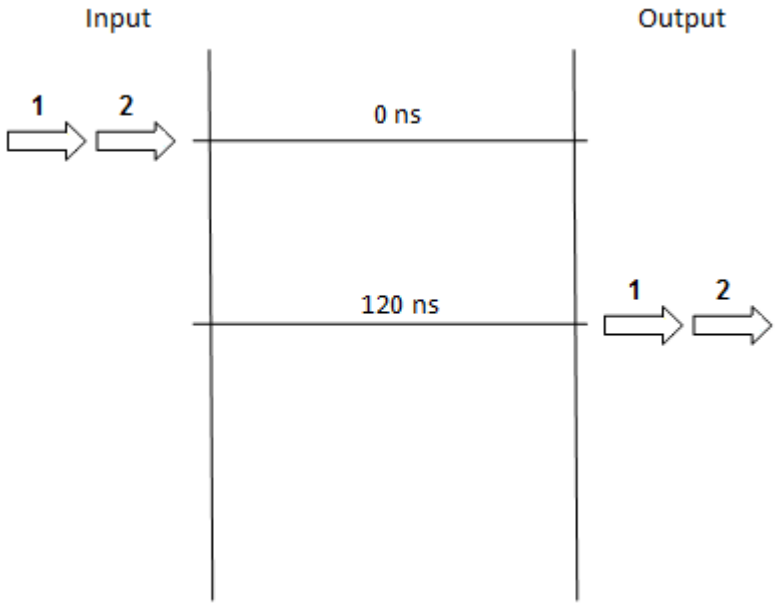

Figure 3-18 Packets times when they enter input ports at the same time

Figure 3-19 shows times when two different packets enter the input ports at different times and exits from output ports. As seen from the figure, if two packets enter different ports of the switch in a 10 nanoseconds time interval, they exit from correct output ports 120 ns after they enter the switch. Note that clock cycle for the switch is 10 nanoseconds.

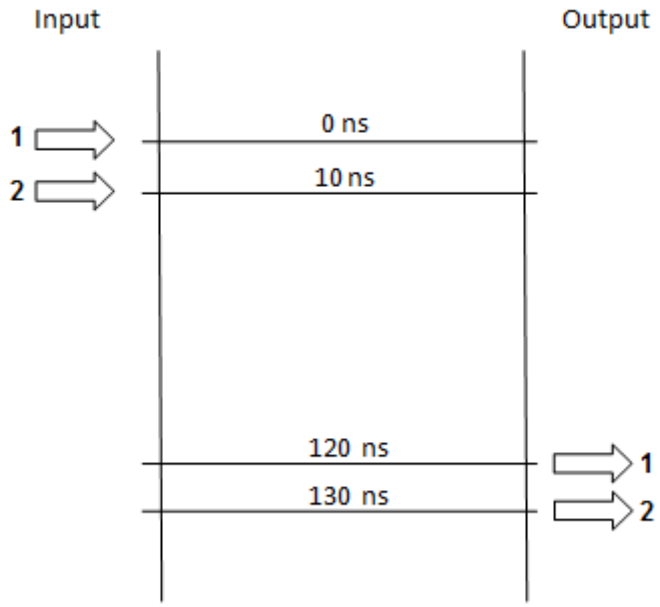

Figure 3-19 Packets times when they enter input ports at different times

It can be concluded that processing time of the switch is 12 clock cycles for any clock cycle interval. Waiting time in the queue because of processing does not change with time when a packet enters the switch. Therefore, it is fixed and independent of any parameter.

The input-queued switch architecture with *i*Slip algorithm emits packets from outputs at 60 nanoseconds time intervals when it works with a clock with period 10 nanoseconds as mentioned in Section 3.2., but the Byte-Focal switch is able to send packets from outputs at 10 nanoseconds time intervals. In order for two architectures to work together properly, we have to set their working speed so that the architectures can extract output packets compatible with each other. Since the Byte-Focal architecture packets exits from output ports 6 times faster than the input-queued switch with *i*Slip, the Byte-Focal switch is arranged to work with a clock of 6 times slower than the input-queued switch with *i*Slip.

A simulation screen is seen in Figure 3-20. As seen from this figure, the departure times for outgoing packets is arranged so that departure time between two consecutive packets is 60 nanoseconds when clock cycle is 10 nanoseconds.

Figure 3-20 The Byte-Focal switch simulation screen

Also number of used resources for different number of ports is given in Table 3-2.

Table 3-2 Resources used in the Byte-Focal switch architecture

	The Byte-Focal Switch	
	4Port	8Port
Number of Slice Registers	11491	76554
Number of Slice LUTs	21139	141264
Number of Block RAM/FIFO	96	640

3.5 Implementation of the Combined Switch Architecture

Since the input-queued switch with *i*Slip algorithm and the Byte-Focal Switch are implemented correctly, the two switches are tested in order to verify the operation of switches. No problems are encountered during tests of both architectures. Therefore, we are confident of the correct operation of the two switches. Test procedures are mentioned in CHAPTER 4 in detail.

After two switch fabrics are tested, we tried to implement the combination of two switch architectures in FPGA. Although two architectures work properly alone, we described some problems which may occur when the combined switch architecture is implemented. These problems are explained below:

1. Although the Byte-Focal switch has resequencing buffers (RB), the input-queued switch with *i*Slip does not have any buffers at the outputs.

2. Working speed of the Byte-Focal switch architecture is 6 times faster than the input-queued switch with *i*Slip.

3. The input-queued switch with *i*Slip does not have an internal processing delay, while the Byte-Focal switch has a processing delay of 12 clock cycles.

Before we start to implement the combined architecture, we have to solve described problems above and propose some methods in order to work two architectures coherently. Proposed solutions are listed below according to each related problems.

1. The Byte-Focal switch has reseqeuncing buffers at the outputs in order to solve the out-of-sequence problem. There are a total of N2 queues for each output ports. Output packets exits from output ports after selection of a packet between packets in N2 queues. Therefore, when the two switches are combined to work together, the packet selection structure should make selection between exiting packets at last stage of each outputs. In other words, packets of the Byte-Focal architecture are selected after exiting the related output port.

2. In order to solve working speed difference problem, the two architectures are arranged so that different clock signals are applied to the Byte-Focal switch and the input-queued switch with *i*Slip. Therefore, two switches do not work with the same clock signal. The combined switch has an internal structure which divides the applied input clock signal into a clock with desired speed.

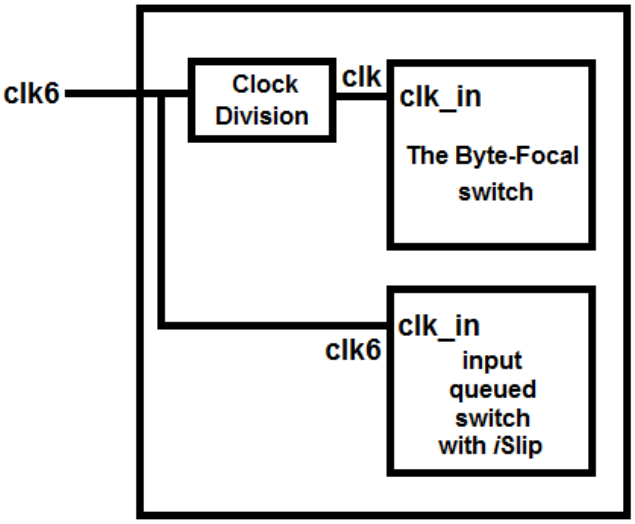

Figure 3-21 Clock division process in the combined switch

Clock division process is shown in Figure 3-21. In the figure, clk6 represents the input clock which is applied to combined switch and also to the input queued switch with *i*Slip. After the clock signal is divided, it is applied to the Byte-Focal switch. Therefore, two architectures are arranged so that output packets exit synchronously.

3. Since we desire a combined system working coherently, two switches should work in synchronous with each other. Therefore, we added same processing delay to the Byte-Focal switch with the Input Queued Switch with *i*Slip so that packet transitions from the output ports starts simultaneously. The processing delay is not considered in performance evaluation, only queuing delay is calculated since processing delay is not significant with respect to queuing delay. Performance evaluations in [1] and [2] are shown in terms of average queuing in which the Byte-Focal and *i*Slip architectures are discussed. Queuing delay may also be excessive and undetermined for different loads. Therefore, if there were real fixed size cells which exist in IP networks, the processing delay is negligible because an actual packet consists of large number of cells. In this situation, queuing delay will be dominant. We started the implementation of the combined switch architecture after we have solution proposals to described problems above. The implemented combined switch architecture is shown in Figure 3-22.

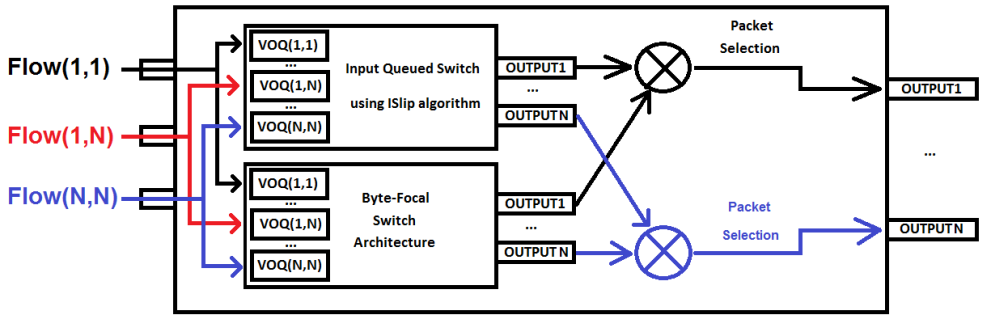

Figure 3-22 Implemented combined switch architecture

As seen from Figure 3-22, virtual output queues of input queued switch with *i*Slip algorithm and the Byte-Focal switch are used together so that input packet is written to buffers of both architectures.

When the packets are written to input buffers, empty signals of buffers are asserted low. Both architectures begin to run their algorithms in order to forward incoming packets to their output ports. Packets begin to exit from outputs after a specific interval of time.

A packet selection module is connected to output ports of both architectures. It makes a selection between output ports of the switches in order to forward the newer packet between the two candidate packets. There is a comparator connected to output ports of Byte-Focal and *i*Slip architectures. This selection is made by selecting the most recent packet between the two alternative output packets. Since the most recent packet is selected by the packet selection module, there is no possibility that a packet which is released from that output before exits. At this point, we should also mention on an important point. If a packet which already exited from an output port of any switch fabric is deleted in the other switch fabric, there would be no change in the overall combined switch fabric since the combined switch extracts output packets of the faster switch under traffic at that instance. Also, this is out of scope of our analysis.

We know that the Byte-Focal switch performance is better for high loads (for loads 0.8 or above) and the input queued switch with *i*Slip has lower queuing delays under low loads (for loads 0.8 or below) for uniform input traffic. Therefore, we expect from the combined switch to have the behavior of the Byte-Focal switch under high loads and the behavior of the input queued switch with *i*Slip under low loads when uniform i.i.d. traffic model is applied.

Our results in Chapter 4 confirms that we have better average delay values compared to the input queued switch with *i*Slip and the Byte-Focal switch but the combined switch uses more FPGA resources.

Table 3-3 shows total FPGA resource used by the combined switch architecture.

Table 3-3 Resources used in the combined switch architecture

	The Combined Switch	
	4 Ports	8 Ports
Number of Slice Registers	14846	84732
Number of Slice LUTs	24608	152186
Number of Block RAM/FIFO	104	672

Total used FPGA resource is approximately equal to sum of resources used by the input queued switch with *i*Slip and the Byte-Focal switch individually. Used FPGA resource by the combined 4-port switch architecture is shown in Table 3. These values are more than total used resources by the two architectures.

Figure 3-23 shows a simulation screen of the combined switch. Packets exit from the output ports at 60 nanoseconds time intervals.

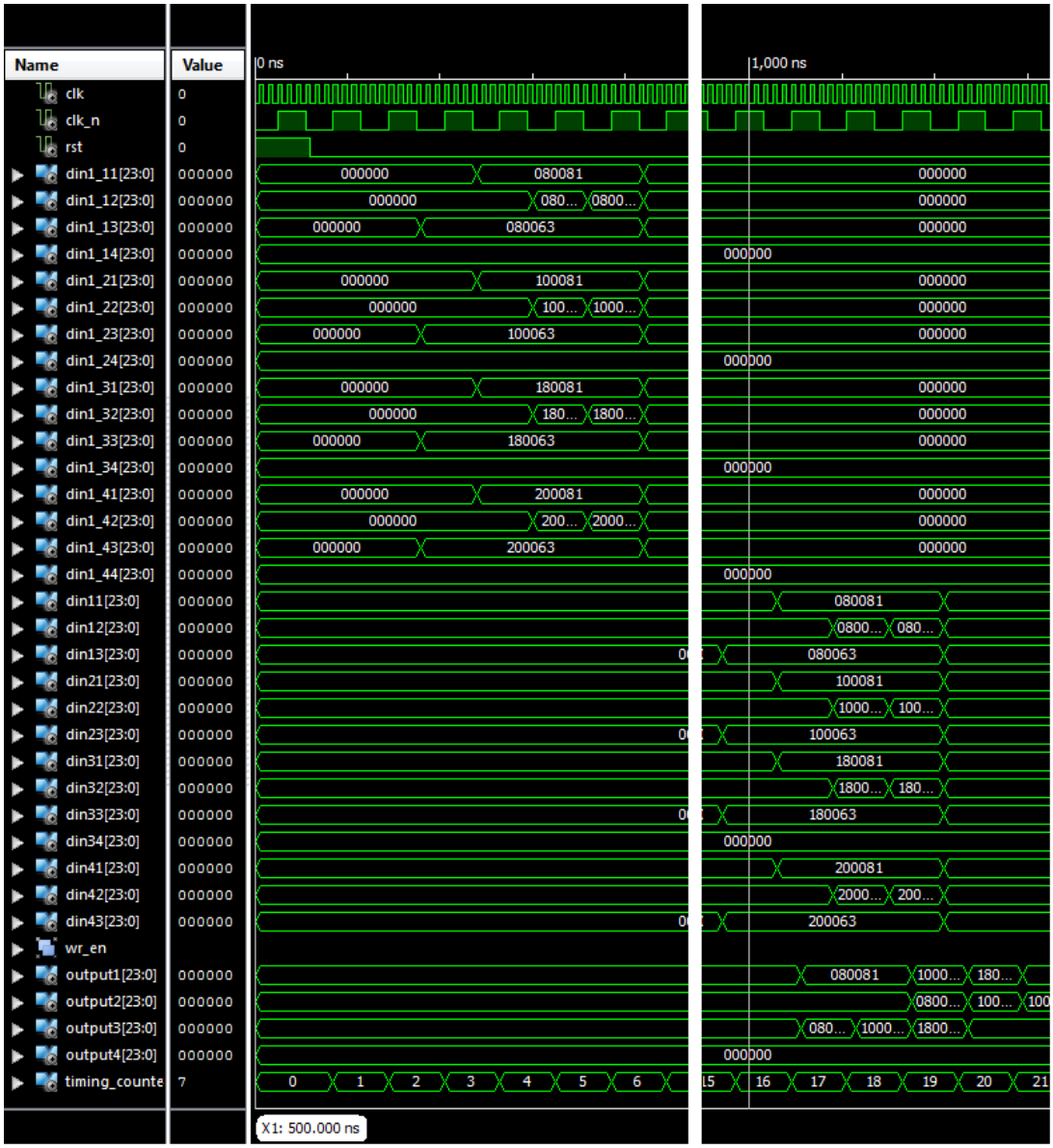

Figure 3-23 Combined switch simulation screen

CHAPTER 4

PERFORMANCE EVALUATION

4.1 Input Traffic Generation

In this section, 4-port and 8-port switch fabric architectures are evaluated in order to compare their performances. Their average delay values are calculated for switches having different number of ports and by applying input traffic with different loads. In order to evaluate switches for different loads, we should be able to generate appropriate input traffic with desired format and we should have a proper test setup in order to test and measure delay data correctly.

Packet distribution for the uniform traffic model is as follows:

Uniform i.i.d : $\lambda ik = \lambda/N$, where i is the input port, k is the output port.

We generate input traffic data for *"uniform i.i.d."*. Data format for input packets are described in Section 3.1. There are 5-bit partitions which are reserved for input port and output port information and 14-bit partition which is reserved for time information.

In order to create input traffic, a C# project is generated. The generated C# code uses random number generation property to produce random data. Composed program is able to generate *"uniform i.i.d."* traffic model with desired load.

The used algorithm to compose *"uniform i.i.d."* model is explained as follows:

1. A timing counter counts for every cycle of a packet generation phase so timing value is increased by one initially.

2. Generate a random number between 1and 100.

3. If the generated random number is smaller than or equal to desired load percentage, then generate an input packet. If the generated number is bigger, then do not generate an input packet. When the packet is not generated, 24-bit data which is composed of all zeros is written to output text file.

4. If an input packet is generated then generate a random number between 1 and number of ports N. The random number generated in this step is the output port to which packet will be forwarded.

5. Input port information is converted to 5-bit binary data, timing counter value is converted to 14-bit binary data and input port information is converted to 5-bit binary data.

6. Finally, input port, time and output port information are concatenated and written to output text file as 24-bit data packet.

This process continues until desired number of packets is generated.

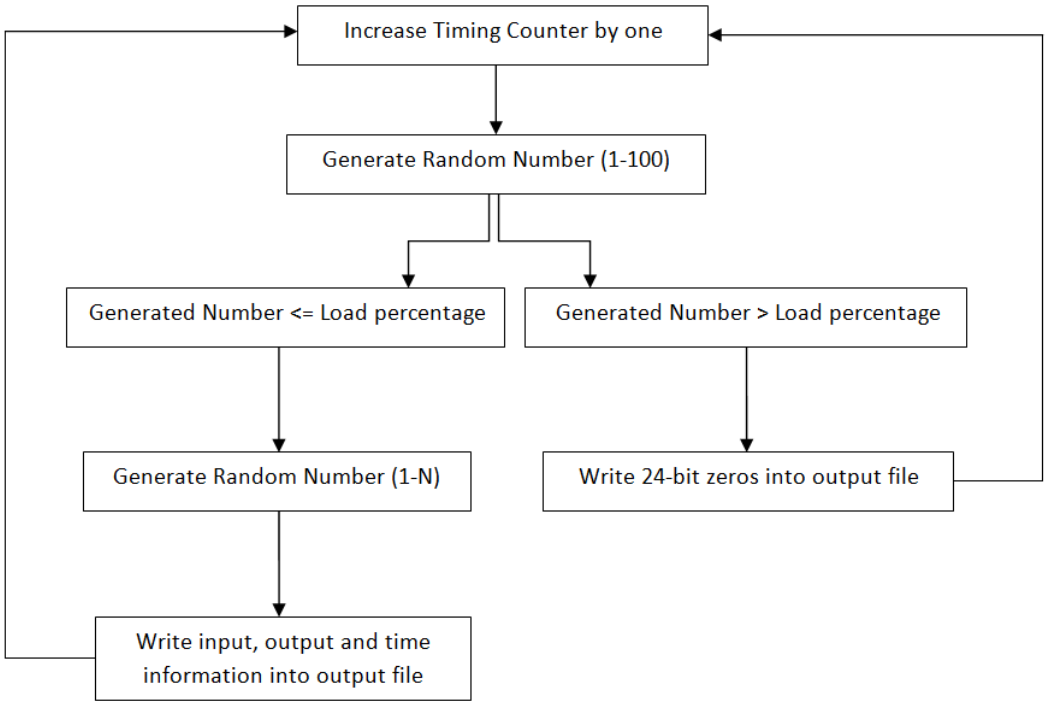

Figure 4-1 Packet generation flow

Packet generation flow is shown in Figure 4-1. For example, desired load is 0.2 (i.e. load percentage is 20%) and N=4 for input port 1. If the number generated between 1 and 100 is larger than 20, generated data will be *"000000000000000000000000"*. Let's say the number generated is 16. Since 16 is smaller than 20, a packet will be generated. Then another random number is generated between 1 and N, let's say the number is 3. If the counter value at that time is 12, the generated packet will be *"000010000000000110000011"*.

Composed C# program is shown in Figure 4-2.

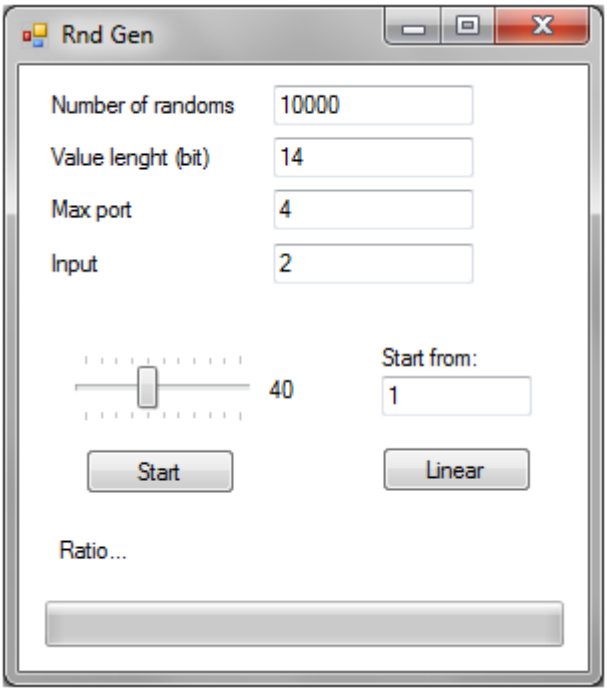

Figure 4-2 Random number generator tool

User enters corresponding data into text boxes and program generates desired data into a text box. In *"Rnd Gen"* program, meanings of text boxes are expained below:

- Number of randoms shows desired number of data.
- Value length is the number of bits which time counter value is converted.
- Max port is the number of ports which switch fabric possesses.
- Input is the number of port for which packets are generated.
- Trackbar (0-100) indicates the load percent of generated traffic.
- Linear button generates input packets which have linear time values while start button generates input packets which have random time values.
- Start from textbox is used to define the initial value of time information.

In *"Rnd Gen"* program, the input packets are composed by entering proper values into textboxes and track bars. For example, if we want to generate input packets with load percentage 40% for input port 2 of a 4-port switch fabric and required packets will have a 14-bit time information starting from 1, we enter the values shown in Figure 4-2 then press *"Linear"* button and input packets are written to a text file.

An example output file of generated packets is shown in Figure 4-3. Load is 0.2 and packets are generated for input port 2. First five bits are assigned for input port information, last five bits are assigned for output port information and bits 6-19 are assigned for time information.

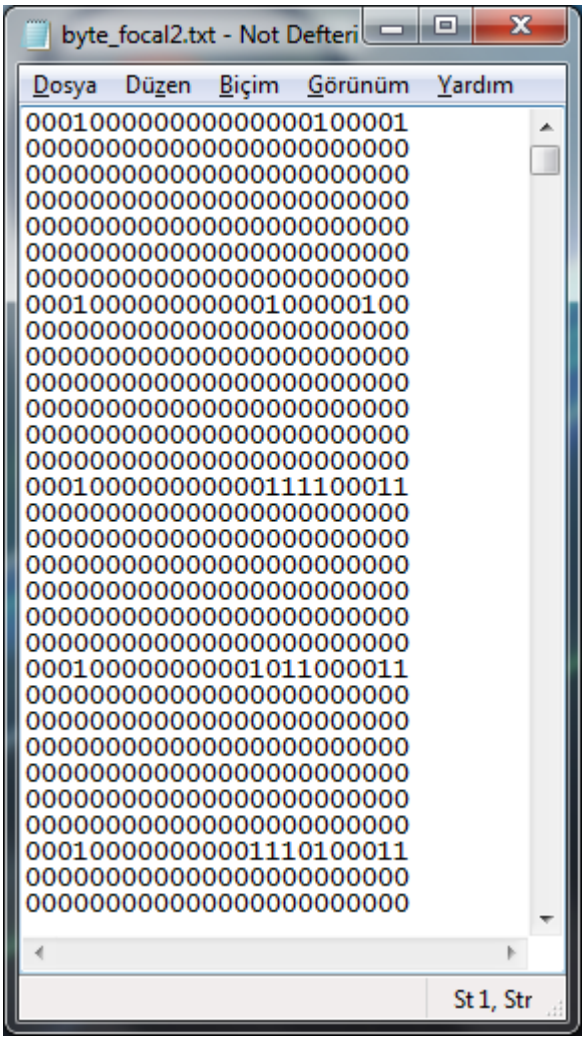

Figure 4-3 Generated input packets for input port 2

4.2 Test Setup

We used VHDL test bench in order to test implemented switch architectures and measure their performance metrics. We have the input queued switch with *i*Slip algorithm, the Byte-Focal switch and the combined switch implementations on FPGA and required input traffic packets are generated until now. Therefore, a proper test setup which is compatible with our implementations and generated packets should be composed.

Xilinx ISE Simulator (ISim) tool is used in order to test the implemented FPGA designs. ISim provides a full-featured HDL simulator integrated within ISE [24].

Testbench tool assignments are given as:

1. Setting timing values for each signal applied to design,
2. Generating essential reset and clock signals,
3. Reading input packets from text files,
4. Sending the input packets which are read from text files to correct input ports,
5. Keeping instantaneous time value by counting a counter value in every clock period,
6. Calculating the queuing delay value by taking difference of times between a packet enters the switch and exits the switch,
7. Writing queuing delay value for every packet into a text file.

Test bench simulator reads an input data in each clock cycle and sends data to related output ports. This operation is performed by a process in VHDL test bench. In this process, a 24-bit line from text file is read. Initially data is checked whether it is composed of all zeros. If it is all zeros then no packet is send to input port. If data is not all zeros, the process checks the last five bits and writes the packet into correct buffer of the input port. An example read process for input port 4 is shown as an example in Figure 4-4.

```
456     --read process
457     i4_reading: process
458     variable rdline4     : line;
459     variable data_read4  : std_logic_vector(23 downto 0);
460     file vector_file4    : text open read_mode is "4.txt";
461        begin
462           wait for 60 ns;
463              while not endfile(vector_file4) loop
464                 readline(vector_file4, rdline4);
465                 read(rdline4, data_read4);
466                    if data_read4 = "000000000000000000000000" then
467                       wr_en1_41   <= '0';
468                       wr_en1_42   <= '0';
469                       wr_en1_43   <= '0';
470                       wr_en1_44   <= '0';
471                       din1_41     <= (others => '0');
472                       din1_42     <= (others => '0');
473                       din1_43     <= (others => '0');
474                       din1_44     <= (others => '0');
475                    elsif data_read4(4 downto 0) = "00001" then
476                       din1_41     <= data_read4;
477                       wr_en1_41   <= '1';
478                    elsif data_read4(4 downto 0) = "00010" then
479                       din1_42     <= data_read4;
480                       wr_en1_42   <= '1';
481                    elsif data_read4(4 downto 0) = "00011" then
482                       din1_43     <= data_read4;
483                       wr_en1_43   <= '1';
484                    elsif data_read4(4 downto 0) = "00100" then
485                       din1_44     <= data_read4;
486                       wr_en1_44   <= '1';
487                    else
488                       wr_en1_41   <= '0';
489                       wr_en1_42   <= '0';
490                       wr_en1_43   <= '0';
491                       wr_en1_44   <= '0';
492                    end if;
493                 wait for 60 ns;
494                       wr_en1_41   <= '0';
495                       wr_en1_42   <= '0';
496                       wr_en1_43   <= '0';
497                       wr_en1_44   <= '0';
498              end loop;
499                 wr_en1_41   <= '0';
500                 wr_en1_42   <= '0';
501                 wr_en1_43   <= '0';
502                 wr_en1_44   <= '0';
503              wait;
504     end process;
```

Figure 4-4 Read process for input port 4

When the packets exit from output ports, queuing delay for each packet is calculated by taking the difference between timing counter value at that time and time value existing in the packet. In other words, queuing delay (t_q) is calculated as:

$t_q = t_{out} - (t_{in} + t_p)$

where t_{out} is time when packet exit from switch, tin time value when packet enter to switch and tp is processing delay. After calculating the difference, queuing delay (t_q) for each exiting packet is written to an output text file. The writing process for output port 3 is shown in Figure 4-5.

```
568     --write process
569     writing3 : process
570     file outfile3     : TEXT open WRITE_MODE is "out3_byte_focal.txt";
571     variable outline3 : line; --line number declaration
572     begin
573      while true loop --while the file end is not reached.
574       if ((output3_old /= port3_output) and (port3_output /= "00000000000000000000000")) then
575       --write(outline,value(real type),justified(side),digits(natural));
576          write(outline3, output3_delay, right, 14);
577       -- write line to external file.
578          writeline(outfile3, outline3);
579       else
580          null;
581       end if;
582       wait for 60 ns;
583      end loop;
584     wait;
585     end process;
```

Figure 4-5 Write process for output port 3

An example output text file in which queuing delay values for each packet is written is shown in Figure 4-6. Queuing delay for each packet is a 14-bit number in terms of clock cycles. For example, fifth packet in the *"out1_islip.txt"* file has a queuing delay of 2 clock cycles.

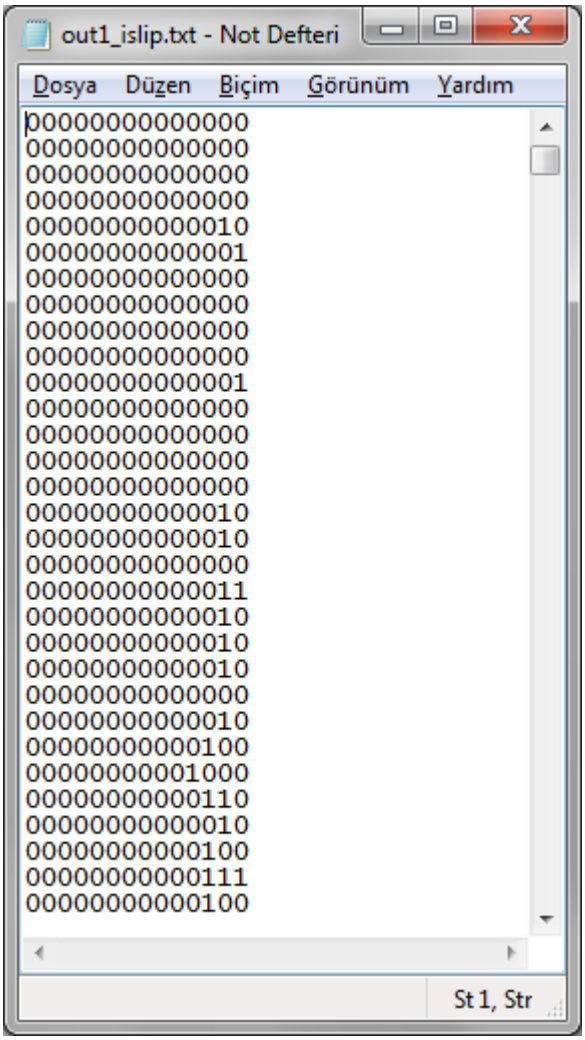

Figure 4-6 Queuing delays for output 1 packets

4.3 Performance Metrics

Performance metrics are described in Section 2.3 in detail. Switch performance is evaluated in terms of average delay, switch size and input traffic load in Shen *et al.* [2]. Since the Byte-Focal switch and input-queued switch with *i*Slip both have 100% throughput, we can say that the combined switch architecture will have also 100% throughput provided that the switch has enough buffers in virtual output queues.

We evaluate the Byte-Focal switch, input-queued switch with *i*Slip and the combined switch in terms of average delay, switch size and input traffic load to be able to compare our results with results in Shen *et al.* [2].

4.4 Simulations

The Byte-Focal switch, input-queued switch with iSlip and the combined switch architectures are simulated using ISE Simulator (ISim) tool from Xilinx, Inc. and the behavioral simulation tool is used. Simulation results are obtained for different size switches, different input traffic types and load values and different architectures.

Average delay-Load graphs are shown below for 4-port and 8-port iSlip, Byte-Focal and combined switch architectures. Average queuing delay values are given in terms of clock cycles and processing delays are not included in average delay calculations. Load is defined as the percentage of total incoming packet density in time. Average delay values are calculated for delay average of all the output ports. Note that, average delay values seen on the graphs are given for queuing delay parts for each fabric, so processing delay values which are given in Chapter 3 would be negligible if a packet consists of a number of cells, for example 48 Byte packets for Cisco routers [27].

In Figure 4-7 and Figure 4-8, the average queuing delay values for iSlip, Byte-Focal and combined switches are drawn on the same graph.

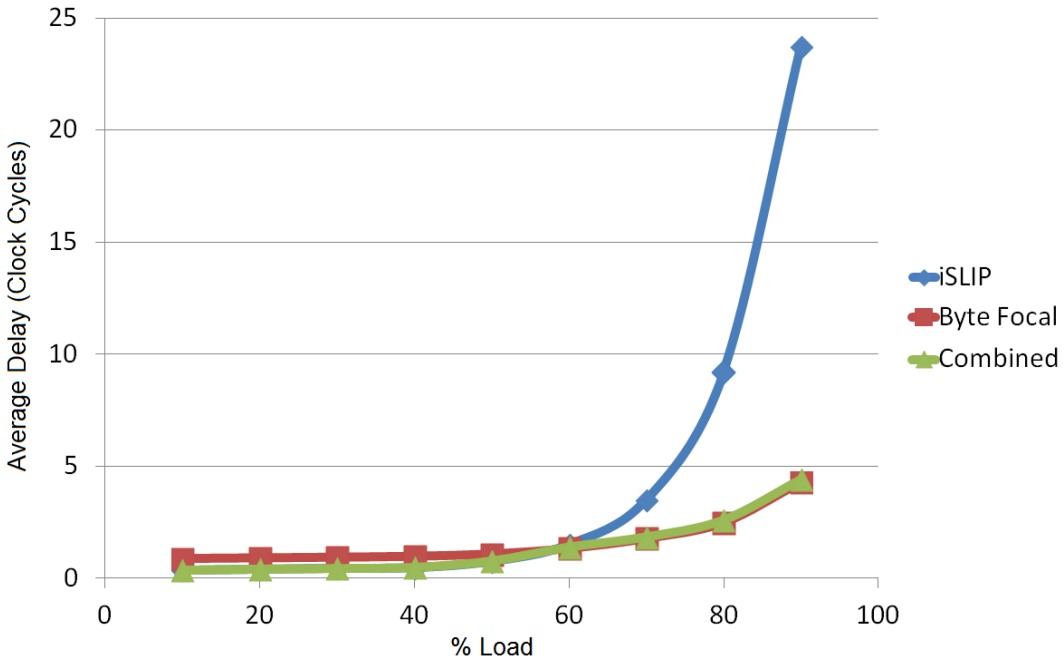

Figure 4-7Average Delay vs. Load for 4-port iSlip, Byte-Focal and Combined Switch under uniform traffic

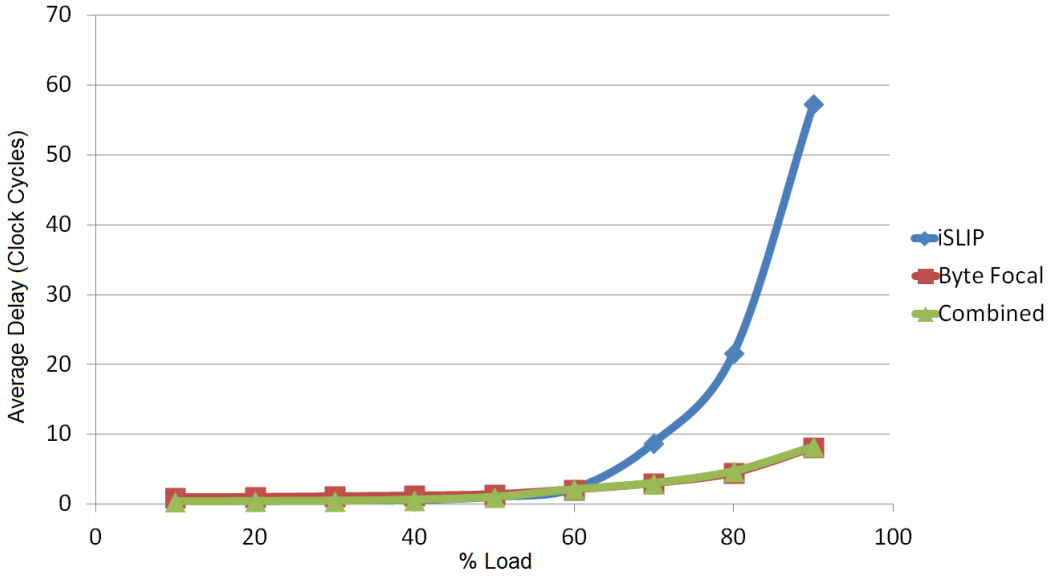

Figure 4-8Average Delay vs. Load for 8-port iSlip, Byte-Focal and Combined Switch under uniform traffic

When the graphs are investigated, we see that for loads below 60% *i*Slip has better performance in terms of average queuing delay and the Byte-Focal switch has lower delay values for loads above 60%. Actually, the plots have expected lines since similar results can be seen from Shen *et al.* [2].

The combined switch has always lower queuing delay values because it includes both iSlip and Byte-Focal switch and a packet selection mechanism selects the faster packets between outputs of the two switch under all traffic loads.

4.5 Results and Discussion

In this chapter, simulations and tests of our proposed combined switch architecture are done. Simulation results are obtained for 4-port and 8-port architectures of the Byte-Focal switch, the input-queued switch with *i*Slip algorithm and the combined switch under different loads and different types of traffic. Average delay - load graphs are given in Section 4.4.

When the obtained average delay – load graphs are investigated, the input-queued switch with *i*Slip algorithm has lower average delay values uniform traffic with loads less than 0.6. However, The Byte-Focal switch has better average delay values for loads higher than 0.6 under uniform traffic. The combined switch has always lower delay values when compared to *i*Slip and Byte-Focal architectures. However, used FPGA resource by the combined switch is slightly more than sum of resources used by *i*Slip and Byte-Focal switches.

CHAPTER 5

CONCLUSIONS

High-speed switch fabrics are needed because of increasing bandwidth demand on today's network applications. In the literature, there are lots of architectures and algorithms proposed so far. Some important algorithms are DRR, PIM and *i*Slip. There are also some important structures proposed such as input-buffered, clos network, crossbar, load-balanced and Byte-Focal architectures.

*i*Slip architecture is widely used since it has low queuing delays, 100% throughput and it is highly scalable. The Byte-Focal architecture is in the structure of a load-balanced switch and it has more performance under heavy loads. When their average queuing delay-load graphs are investigated in [2], it can be seen that *i*Slip delay is lower than the Byte-Focal under uniform traffic with low loads. However, the Byte-Focal switch has better performance than *i*Slip under high loads. Therefore, the two fabric architectures have different behaviors under different traffic loads. Therefore, the combined switch idea works well with all traffic loads.

In this thesis, we implement *i*Slip and Byte-Focal switch architectures in FPGA and verify them by doing simulations. When the two switches are implemented, combined switch architecture is composed by implementing the *i*Slip and Byte-Focal architectures together in FPGA and working them in parallel. After implementation of combined switch, simulations are done for *i*Slip, Byte-Focal and combined architectures and the results are given in Chapter 4. According to simulation results, advantage of the combined architecture is that it has always lower delay values of *i*Slip and Byte-Focal architectures and it has 100% throughput under uniform and hot-spot loading. Also, the combination of switches is a novel idea since there is no reference combined architecture in the literature. A similar approach is reconfiguration in runtime for QoS increase [5], [16]. On the other hand, both *i*Slip and Byte-Focal switches are located in FPGA in order to implement the combined architecture and more resource for the combined switch is used than used for a single architecture.

We used round robin scheduling of the Byte-Focal switch because of its relatively low complexity in our implementations. We expect a better performance especially under high input traffic loads if the other scheduling algorithms such as LQF, fixed threshold or dynamic threshold for Byte-Focal switch [2] are implemented.

In this thesis, *i*Slip, Byte-Focal and combined switch architectures are implemented on FPGA and they are simulated under different switch size and different traffic load. In the future, the added delay to *i*Slip switch for the synchronization might be removed by proposing a new solution. Furthermore, the parts which have the same functions of the two architectures might be shared in order to reduce the total resource consumption. This can be done by modular implementation of the common parts which is adaptable for both switch fabrics.

REFERENCES

[1] McKeown, N., *The iSlip Scheduling Algorithm for Input-Queued Switches,* IEEE/ACM Transactions on Networking, Vol. 7, No. 2, 1999.

[2] Shen, Y., Panwar, S. S., Chao, H. J., *Design and Performance Analysis of a Practical Load-Balanced Switch,* IEEE Transactions on Communications, Vol. 57, No. 8, 2009.

[3] Keslassy, I., Chuang, S.T., McKeown, N., *A Load-Balanced Switch with an Arbitrary Number of Linecards,* IEEE INFOCOM 2004.

[4] Chang, C. S., Lee, D. S., Jou, Y. S., *Load-Balanced Birkhof – von Neumann switches, part I : one - stage buffering,* Computer Communications, 2002.

[5] Kachris, C., Vassiliadis, S., *A Dynamically Reconfigurable Queue Scheduler*, IEEE, 2006

[6] Audzevich, Y., Ofek, O., Telek, M., Yener, M., *Analysis of load-balanced switch with finite buffers*, IEEE, 2008.

[7] Chang, C. S., Chen, W. J., Huang, H. Y., *Birkhoff-von Neumann Input Buffered Crossbar Switches,* IEEE, 2000.

[8] Yu, C. L., Chang, C. S., Lee,D. S., *CR Switch: A Load-Balanced Switch With Contention and Reservation,* IEEE/ACM Transactions on Networking, Vol. 7, No. 2, 1999.

[9] He, R., Delgado-Frias, J. G., *"Fault Tolerant Interleaved Switching Fabrics for Scalable High-Performance Routers"*, IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 12, 2007.

[10] Hu, B., Yeung, K. L., *Feedback-Based Scheduling for Load-Balanced Two-Stage Switches,* IEEE/ACM Transactions on Networking, Vol. 18, No. 4, 2010.

[11] Rojas-Cessa, R., Dong, Z., *Load-Balanced Combined Input-Crosspoint Buffered Packet Switches,* IEEE, 2011.

[12] Mhamdi, L., *A Partially Buffered Crossbar Packet Switching Architecture and its Scheduling,* IEEE, 2008.

[13] Shen, Y., Panwar, S. S., Chao, H. J., *Performance Analysis of a Practical Load Balanced Switch*, IEEE, 2006.

[14] Minkenberg, C., *Performance of i-SLIP Scheduling with Large Round-Trip Latency*, IEEE, 2003.

[15] Duan, Q., *Quality of service provision in combined input and crosspoint queued switches without output queueing match*, Computer Communication, 2006.

[16] Laskaridis, H. S., Papadimitriou, G. I., Pomportsis, A. S., *Reconfigurable ATM Switch Fabrics Using Traffic History*, IEEE, 2002.

[17] Yoshigoe, K., *Trends in highly scalable crossbar-based packet switch architecture*, Computer Communications, 2008.

[18] Chao, H. J., Liu, B., *High performance switches and routers*, 2007.

[19] Chang, C., Lee, D., Jou, Y., *Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering*, *Computer Communications*, vol. 25, pp. 623-634, 2002.

[20] Keslassy, I., McKeown, N., *Maintaining packet order in two-stage switches*, IEEE INFOCOM, vol. 2, pp. 1032-1041, 2002.

[21] Chang, C. S., Chen, J. W., Huang, H., Y., *On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and Von Neumann*, IEEE IWQoS, 1999.

[22] McKeown, N., Anantharam, V., Walrand, J., Achieving 100% throughput in an input-queued switch, Proceedings of IEEE INFOCOM'96, pp. 296-302, 1996.

[23] http://www.xilinx.com/support/documentation/application_notes/xapp462.pdf (last visited on 12.02.2013)

[24] http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/plugin_ism.pdf (last visited on 12.02.2013)

[25] http://www.eetimes.com/design/communications-design/4143689/Calculating-Speedup-in-Switch-Fabric-Designs (last visited on 12.02.2013)

[26]http://www.cisco.com/en/US/products/hw/routers/ps167/products_tech_note09186a00801e1da7.shtml (last visited on 12.02.2013)

[27]http://www.cisco.com/en/US/products/hw/switches/ps1925/products_white_paper09186a00800887ae.shtml (last visited on 12.02.2013)