SWITCH FABRIC SCHEDULERS WITH INTELLIGENT MULTI-CLASS SUPPORT: DESIGN, IMPLEMENTATION AND EVALUATION ON FPGA

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

BY

MURAT AKPINAR

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

SEPTEMBER 2014

Approval of the thesis:

SWITCH FABRIC SCHEDULERS WITH INTELLIGENT MULTI-CLASS SUPPORT: DESIGN, IMPLEMENTATION AND EVALUATION ON FPGA

submitted by **MURAT AKPINAR** 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. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

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

Examining Committee Members:

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

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

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

Dr. Nizam Ayyıldız
REHIS-TTD-AGTM, ASELSAN _____

Dr. Ali Erkin Arslan
MGEO-ASMD-IUSTM, ASELSAN _____

Date: _____

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: MURAT AKPINAR

Signature :

ABSTRACT

SWITCH FABRIC SCHEDULERS WITH INTELLIGENT MULTI-CLASS SUPPORT: DESIGN, IMPLEMENTATION AND EVALUATION ON FPGA

AKPINAR, Murat

M.S., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Şenan Ece Schmidt

September 2014, 102 pages

The applications in the contemporary computer networks require end-to-end Quality of Service (QoS). Moreover, different applications have different QoS requirements. Thus, it is important to support QoS in the network layer routers which can be achieved by scheduling the output queues in output queued routers. However, pure output queued routers are not easy to build. Hence, it is important to equip the fabric schedulers of input queued switches with QoS support. Thus, it is an important research problem to support QoS in input queued routers. In this thesis we investigate the VOQ fabric scheduler algorithms. Better QoS support for different applications is possible by implementing per flow queues at the input ports rather than coarse virtual output queues per output port. The first contribution of this thesis is an intelligent multi-class (IMC) VOQ architecture which is independent from fabric scheduler algorithms. Additionally, 2 different algorithms are proposed for intelligent side of the IMC VOQ architecture. The second contribution is a modular hardware design for fabric schedulers that support multi class. The design is carried out on FPGA by implementing the well-known ISLIP together with the proposed IMC unit. The correctness of the operation of the designed hardware is verified by comparing to a software simulator. The thesis further presents discussions of implementing other scheduler algorithms using the same hardware architecture and its scalability. The thesis presents the evaluation of FPGA resource usage of proposed IMC VOQ iSLIP.

Keywords: FPGA,iSLIP,QoS,VOQ,IMC VOQ,Switch Fabric,Fabric Scheduler

ÖZ

AKILLI ÇOKLU SINIF DESTEGİ OLAN ANAHTAR ÖRGÜSÜ ÇİZELGELEYİCİLERİNİN TASARIMI, FPGA ÜZERİNDE GERÇEKLEŞTİRİLMESİ VE DEĞERLENDİRİLMESİ

AKPINAR, Murat
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Şenan Ece Schmidt

Eylül 2014 , 102 sayfa

Bilgisayar ağlarında günümüzde kullanılan uygulamalar ağ iletişimi kalite servisine ihtiyaç duymaktadır. Farklı uygulamaların farklı kalite servisi ihtiyaçları bulunmaktadır. Bu sebeblerle ağ katmanındaki cihazlarda kalite servisi sağlamak önemlidir. Çizelgeleyiciler yönlendiricilerin kalite servisini doğrudan etkilerler. Çıkış tamponlu çizelgeleyici algoritmaları kalite servisini desteklemelerine rağmen yükselen trafik hızları sebebiyle donanım üzerinde gerçeklenmesi zor hale gelmişlerdir. Bu yüzden giriş tamponlu çizelgeleyici algoritmalarında kalite servisi sağlamak önemli bir araştırma konusu haline gelmiştir. Bu tez kapsamında sanal çıkış tamponlu (SÇT) çizelgeleyiciler incelenmiştir. Giriş portlarında her bir akış için farklı tampon uygulamak genel sanal çıkış tamponlu yapılara göre daha iyi kalite servisi sağlayacaktır. Bu tezin ilk katkısı çizelgeleyici algoritmalarından bağımsız çalışan akıllı çoklu sınıflı (AÇS) sanal çıkış tamponlu yapıdır. Ayrıca bu yapı için iki farklı algoritma önerilmiştir. Bu tezin ikinci katkısı çoklu sınıf desteği olan çizelgeleyiciler için sunulan modüler donanım tasarımıdır. Önerilen AÇS birim sıkça kullanılan iSLIP çizelgeleyicisi ile birlikte FPGA üzerinde gerçeklenmiştir. Bu tasarımın doğruluğu bir yazılım simülatörü ile karşılaştırma yapılarak kanıtlanmıştır. Bu tez ayrıca farklı çizelgeleyicilerin, gerçeklenmiş olan donanım mimarisi kullanılarak nasıl gerçeklenebileceğini ve bu mimarinin ölçeklenebilirliğini tartışmaktadır. Önerilen AÇS SÇT iSLIP tasarımının FPGA kaynak kullanımı da değerlendirilmiştir.

Anahtar Kelimeler: FPGA,iSLIP,Kalite Servisi,Sanal Çıkış Tamponu (SÇT),Akıllı Çoklu Sınıflı (AÇS),Anahtar Örgüsü,Çizelgeleyici

To my family

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Şenan Ece Schmidt who has supported me continuously with her motivation and immense knowledge.

Besides my advisor, I would like to thank the thesis committee: Prof. Dr. Semih Bilgen, Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı, Dr. Nizam Ayyıldız, and Dr. Ali Erkin Arslan for their insightful comments and questions.

I would like to thank ASELSAN for their permissions for attending classes and for providing some technical equipments and software.

I also thank TÜBİTAK for their supports with TÜBİTAK-BİDEB M.s. scholarship.

I am grateful to my friends, Ali Özgün, Burcu Özgün, Ahmet Değirmenci, Erhan Gündoğdu, Alican Yüksel, Fırat Erciş, Savaş Karadağ and Koray Karakurt for their support throughout the development and the improvement of this thesis.

I also thank my mom, dad and brother for their valuable support.

Above all, I thank Gizem Sönmez for her peerless support and I am grateful to every moment she stands by me.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

DRR	Dual Round-Robin
E2E	End-to-End
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
HOL	Head-of-Line
IMC	Intelligent Multi-Class
ipTV	Internet Protocol Television
LQF	Longest-Queue First
PIM	Parallel Iterative Matching
QoS	Quality of Service
VOQs	Virtual Output Queues

CHAPTER 1

INTRODUCTION

The applications in the contemporary computer networks require end-to-end-Quality of Service (QoS). The widest support for E2E QoS at the network layer routers and switches is provided by QoS scheduling at the output ports. Full QoS control can be achieved by output schedulers in switches where all queuing takes place at the outputs. However, for such switches, the hardware must operate much faster than the line speed which increases the implementation cost and prohibits implementing all switches and routers as output queued. Products such as [1] which uses input queueing system architecture achieve throughput that are close to 1 Tbps that employs a slower switch fabric. In such switches a *fabric scheduler* component computes the mapping of inputs and outputs to maximize the switch throughput. Furthermore, the delay at the input queues contributes to the end to end QoS of the packets for input queued switches. Hence, it is an important research problem to support QoS in the fabric schedulers.

There are a number of fabric schedulers in the literature [2][3][4] which employ Virtual Output Queues (VOQs) where each input port has dedicated queues for each output port. The mapping of inputs to outputs is generally carried out with signaling between input and output ports. These fabric schedulers are evaluated according to the metrics of packet delay, throughput and scalability [5]. That is, emphasize has to be given in the hardware because the fabric scheduler gets involved in the data path and it handles each packet. Furthermore, high speed switch and routers are implemented to forward the packets in fixed-size units as in [6]. However, the IP packets are variable in size, and this requires segmentation into fixed-size units and then re-

assembly after processing.

Well-known fabric schedulers such as iSLIP [2] are designed in such a way that each input sends a single request to the output ports for which it has queued packets. However, QoS support for different traffic classes requires installing multiple queues for each output at each input which leads to multiple requests from a given input port to a given output port.

Motivated by the QoS support requirements of fabric schedulers together with their performance requirements, the contributions of this thesis are as follows: An Intelligent Multi-Class (IMC) unit which resolves these multiple requests without changing the original fabric scheduling algorithm is proposed. An algorithm of request selection for the proposed IMC which achieves better packet delay for high priority class is developed. An FPGA component level design, implementation and evaluation for the iSLIP fabric scheduler as an example fabric scheduler with the proposed IMC is carried out. The FPGA design further includes the segmentation and reassembly functions together with a switch fabric. Moreover, generalization of the design is discussed to demonstrate its scalability and its support for different fabric schedulers.

The rest of the thesis is organized as follows:

In Chapter 2, the multimedia services over the internet is explained. The traffic characteristics and the QoS requirements of multimedia services are mentioned afterwards. Additionally, the existent QoS support for multimedia services are discussed. Moreover, the literature overview on fabric scheduling algorithms are mentioned. Here, the iSLIP[7] is discussed in detail. Lastly, our implemented architecture, iSLIP with IMC unit is compared with the existing fabric scheduling algorithms.

In Chapter 3, the proposed IMC VOQ structure, which is independent from the fabric scheduler, is explained in detail. Furthermore, the 2 algorithms that are invented in this thesis for intelligent part of the IMC unit are explained.

The hardware implementation of the iSLIP with IMC unit is analyzed in Chapter 4. Also, the simulation results for the hardware blocks are presented in order to clarify the functionality of the hardware implementation.

In Chapter 5, the generalization of the hardware implementation is discussed. The procedures which must be followed in order to change the priority levels and the number of ports are questioned. Moreover, the modifications that have to be made in order to use a different fabric scheduler with IMC unit except the iSLIP are explained.

In Chapter 6, the composed simulation environment is explained. The verification methods of the hardware implementation is mentioned. The simulation results are presented in terms of the performance metrics such as packet delay and throughput. The hardware implementation resource usage is also told. Furthermore, the simulation results are discussed to clarify whether they are anticipated or not.

The thesis concludes with Chapter 7, where summary and future direction of our research are presented and discussed.

CHAPTER 2

RELATED WORK

2.1 Multimedia Services over The Internet

Sharp increase in the number of the Internet users results also in an increase in the internet traffic. Moreover, today networks support different communication modes such as video, voice and data that provide new applications for various activities including entertainment, shopping, etc. [8]. According to Cisco Systems Inc., consumer entertainment and communications must offer personalized media and interactive ipTV services which combine entertainment, communications, and the Internet [9]. Considering that the existing network structures reach their limits in order to support these new IPTV services. That is, networks must be scalable to high number of customers, increase bandwidth utilization, and provide quality of service (QoS) so that they can meet the requirements of such services. Thus, the design of routers/switches has become an interesting topic.

2.1.1 The Traffic Characteristics and QoS Requirements of Multimedia Services

Multimedia services increase the bandwidth requirement. A consumer who has multimedia services such as ipTV needs higher bandwidth than a consumer only receiving high speed Internet services. The video is delivered to the subscriber's set-top-box by the service provider. The quality of the service is controlled by that service provider which determines the encoding rate. For instance, MPEG2 compression standard consumes almost 3.75 Mbps [9]. On the contrary, newer compression standard MPEG4

consumes 2 Mbps in order to supply the same service quality [9]. While servicing TV-broadcast, bandwidth consumption depends on both the encoding rate and the number of channels offered. For example, providing a service of 200 channels of MPEG2 requires almost 750 Mbps of bandwidth. On the other hand, while there is the video on demand service, total amount of bandwidth depends on both the encoding rate and the number of the viewers.

In order to satisfy user's experience, network-related QoS requirements of multimedia services such as packet loss rates, jitter, delay, and network congestion must be taken into consideration [10]. The video over IP streams are degraded by the packet loss. Set top boxes are able to handle with a packet loss which does not last more than a second [9]. In this manner, the packet loss must not exceed the value which can be handled by the set top boxes to improve the user's quality of experience. The jitter, furthermore, is another important parameter which can affect the user. The jitter value must be kept lower than almost 150 ms which can be compensated by the set top boxes [9]. Although absolute delay is not important for video delivery, the rest of the services care about the delay parameter. Consider that, multimedia services can also be served together with voice over IP (VoIP) and other real-time traffic. Thus, reliable scheduling mechanisms are required with different queuing strategies. Moreover, video-related QoS challenges such as minimization of time of channel change must be dealt with in order to increase quality of experience [10]. The channel change time affects subscriber satisfaction about the service; therefore, network solution should minimize it[9].

2.1.2 Current QoS Support for Multimedia Services on Transport and Application Layers

There are various proposals carried on the transport layer and the application layer which are above the network layer, in order to meet the requirements of the multimedia services.

In the article of Murcia [11], it is stated that better QoS results for ipTV are obtained with an automated RSVP-TE LSP reservation mechanism and by adding some video-awareness "intelligence" to defined Label Switching Routers in the MPLS core

network. The proposed approach is also implemented on Juniper routers with the help of JUNOS SDK in order to test the results.

Furthermore, Luebben [12] suggests a method for fast rerouting of IP multicast traffic during link failures in managed ipTV networks that is possible by making minor modification to the current multicast routing protocol (PIM-SM).

Content and overlay-aware scheduling for peer to peer streaming, which is based on the frame importance feature of the compressed video, is offered in [13]. In that scheduling, the most important data is initially sent to the entire network. Also, frequently switching peer connections is preserved with the aim to use the network resources better.

In order to maximize the end-users' quality of experience, in [14] the Quality-Oriented Adaptation Scheme (QOAS) is proposed for delivering multimedia streams. Depending on the feedback of the clients about the service quality they received, the server is forced to make dynamic changes to transmitted streams.

2.1.3 Buffer Management and QoS Scheduling as QoS Support at The Network Layer

While there is the increasing speed in network applications, QoS requirements should still be satisfied on the network layer. The solutions which are implemented on higher layers such as the transport and application do not give satisfactory results when they are compared with the solutions on the network layer.

Additional to the solutions based on the transport and application layers, further developments are proposed to be implemented on the network layer. The buffer management techniques and QoS scheduling algorithms for output queued switches can, for example, be preferred so that the QoS requirements of the multimedia services can be fulfilled.

A method, suggested in [15], can provide delay guarantees under a traffic contract. In this case, the approach is based on the mixture of buffer management and output QoS scheduling mechanisms. In the output network interface card (NIC), the incoming

packets are separated according to the type of service(ToS) and are stored in different queues. The real-time packets which have QoS guarantee with traffic contracts are stored in conformant queue, whereas the rest of the real-time packets are stored in excess queue. In this excess queue, the buffer management protocol is applied which removes the expired excess packets. The rest of the traffic, however, is stored in the best-effort queue. As an addition to buffer management, while scheduling, the queues which consist of real-time traffic packets are prioritized.

About the issue, Bai [16] proposes an application aware buffer management scheme. In the compressed video, like MPEG video, the frames can be differentiated in accordance with their importance as the I, P and B frames in descending order [16]. Although the loss of I frames affect the image quality of entire Group of Picture (GoP), the loss of less important B frames partially influence the quality of a single frame [13]. Depending on this characteristic of compressed video, it is aimed to increase the loss tolerance for a video service in [16]. By assessing the importance of the video frames, higher level video quality can be acquired with the same packet loss ratio.

The current support on the network layer is mostly QoS scheduling for output queued switches. For those output queued switches, many algorithms have been proposed to supply satisfactory QoS guarantees [17]. Parekh and Gallager succeeded in reaching hard delay bounds by implementing Output Queued (OQ) [18] packet-switch using a Weighted Fair Queuing (WFQ) scheduling algorithm in the IP router [19] [20]. On the other hand, it is inapplicable for large switches since an OQ packet-switch needs an internal speedup [21]. Moreover, when the speed of a single input port increases to gigabits per second, an OQ packet-switch is not scalable anymore [22].

Additional to QoS scheduling and buffer management techniques, the QoS support can be applied on fabric schedulers. These works which are implemented on fabric schedulers is mentioned in Section 2.2.

2.1.4 Commercial Solutions for QoS Support

As a leading company in the network systems area, Cisco proposes enhancements about delivering multimedia services.

The Cisco Enterprise Content Delivery System (ECDS) is developed in order to support video and media delivery over the existing WAN architecture [23]. Cisco ECDS can be implemented together with Cisco Wide Area Application Services(WAAS) to acquire a complete Wide Area Network(WAN) solution which is optimized for both video and data. Cisco ECDS is a media delivery platform which consists of the Service Engine, the Content Delivery System Manager (CDS Manager) and the Service Router. The service engine is responsible for edge content streaming, caching and download to endpoint devices such as PCs. Live and video on demand(VoD) are supported by content streaming in the format of H.264, Adobe Flash, Windows Media and Apple QuickTime. In contrast, the service router manages the requests from the endpoint clients. Depending on the load conditions and location of the endpoint client, the most available service engine is arranged by the service router. Ultimately, the content delivery system manager supplies a graphical interface in order to control the elements of a Cisco ECDS network.

Additional to Cisco ECDS, Cisco has developments inside the router in order to prioritize the video services. Cisco I-Flex design is supported by Cisco routers such as XR12000 and 7600 series. Cisco I-Flex design combines shared port adapters(SPAs) and SPA interface processors(SIPs). In that way, the service prioritization for data, voice and video services are offered together with a modular architecture [24]. Those modular adapters and interface processors can be changed across the Cisco carrier routing platforms such as Cisco 7304 Router, Cisco 7600 Series Routers, Cisco 12000 Series Routers and Cisco XR 12000 Series Routers. Additional to Cisco ECDS, Cisco has developments inside the router in order to prioritize the video services. Cisco I-Flex design is supported by Cisco routers such as XR12000 and 7600 series. Cisco I-Flex design combines shared port adapters(SPAs) and SPA interface processors(SIPs). In that way, the service prioritization for data, voice and video services are offered together with a modular architecture [24]. Those modular adapters and interface processors can be changed across the Cisco carrier routing platforms such

as Cisco 7304 Router, Cisco 7600 Series Routers, Cisco 12000 Series Routers and Cisco XR 12000 Series Routers.

2.2 Fabric Scheduling

Quality-of-service (QoS) is an important focus of designing routers and switches since many applications such as voice over IP and IPTv need quality-of-service (QoS) guarantees including throughput, packet delay and jitter [17]. For output queued switches, many algorithms have been proposed which supply satisfactory QoS guarantees. However, they are not suitable for commercial use since output queued switches are not preferable due to their high cost. Thus, fabric scheduling algorithms which gives QoS guarantees become more significant for the input queued switches.

2.2.1 Overview of High-speed Router Operation

The router's functions can be categorized into two: datapath functions and control plane functions [25].

The functions which are performed on every packet passing through the router are called as datapath functions. The forwarding decision, forwarding through the switch fabric and scheduling decision are datapath functions that are processed for each packet. A packet is firstly welcomed by the forwarding engine. The destination IP address is masked by subnet mask by implementing logical AND operation. The resulting masked address is searched in the forwarding table. By looking up the forwarding table,the destination output port is decided. Additional to the forwarding decision, classification is also implemented. Depending on this classification, the packet may be discarded or it's priority level may be changed. Afterwards, time-to-live (TTL) value is decreased and a new header checksum is computed. Moreover, fabric scheduler decides which input ports and output ports are connected, and then according to that decision, packets are forwarded through switch fabric on the line speed[25]. Additionally, the datapath functions must be implemented in hardware since they are performed in a very short time corresponding the packet transmission time. For example, a 64-byte packet's transmission lasts 51ns at 10Gbps. Thus, it can

be said that only hardware implementations can meet such though time constraints [26].

The control plane functions are not performed as frequent as the datapath functions. Those control plane functions are the system configuration, management and exchange of routing table information. In the system, the router controller is the one that manages the routing table by exchanging the topology information with other routers. However, there is no strict time limitation for the control path functions. Thus, the control path functions are mostly implemented in the software.

2.2.2 Switch Fabric Scheduling Problem Definition

A packet switch is a multiple input-multiple output device that transfers the packets at the input ports towards the output ports according to their destination information. To this end, the switch fabric provides one to one interconnection between input and output port pairs. It is possible to see that multiple inputs receive packets that are destined to the same output resulting in contention which are resolved by implementing buffers at the input ports. Such switch architectures are called input queued (IQ). Similarly, switching multiple packets that contend for the same output by increasing the interconnection bandwidth (fabric speed-up) is another approach. When the switch fabric is operating faster than the input line speed, the excess amount of packets that are switched must be buffered at the output ports, and this brings about an output queued (OQ) switch. A switch with N input and output ports (NxN) with a speed-up of N is a pure OQ switch without any buffers at the inputs which can guarantee 100% throughput [25] and enable precise Quality of Service control on the traffic flows. However, the implementation of such switch fabrics does not scale to high line speeds and large number of ports [22]. An *NxN* switch with a fabric speed-up of $1 < S < N$ require buffers at both inputs and outputs constituting a combined input output queued (CIOQ) switch [27].

The buffers at the input ports for IQ and CIOQ switches are organized in virtual output queues (VOQ). In the VOQ buffering, each input i, $i = 1 \ldots N$ has N logical queues $\mathrm{VOQ}_{i,j}$, $j = 1 \ldots N$. $\mathrm{VOQ}_{i,j}$ stores the packets that arrive at input i and are destined to output j. Hence, the packets that arrive at a given input port that are destined to

different output ports are stored in separate logical queues and they do not block each other.

Furthermore, the switch fabric has to be dynamically configured for each packet transfer cycle by a fabric scheduler. Here, a fabric scheduling problem can be thought as a bipartite graph matching problem [25].

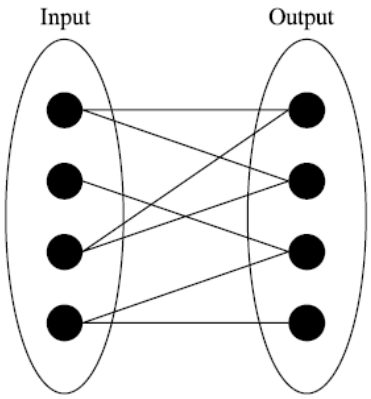

Figure 2.1: Bipartite graph matching example.

In the Fig.2.1 , the nodes on the left side represent input ports, and the nodes on the right side represent output ports. The edges between the input and output ports imply that there are requests from the input port to the output port since there is a packet in the input port which is detonated to related output port. At most, there can be N^2 edges. A scheduler is required to select a set of matches by considering that each input can be connected to at most one output and each output can be connected to at most one input. Such matching of input-output can be represented as a permutation matrix $M = (M_{i,j}), i, j \leq N$ where $M_{i,j} = 1$ if input i is matched to output j in the matching process.

A packet switching from input ports to output ports is a high-speed data plane operation. In this case, the common practice is implementing the switch in fixed size units as in Cisco Cells [28]. However, the router receives variable size IP packets on its input ports, and forwards them on its outputs. Hence, the fragmentation of the IP packets into fixed size units before switching, and the reassembly of the fixed size units back into variable size IP packets are required.

2.2.3 Performance Metrics

While designing a fabric scheduler; several factors, called as performance metrics, must be considered:

- *Throughput and Delay.* The algorithm should supply high throughput with low delay. In other words, the input and output ports should be matched as many as possible in each time slot [25].
- *Fairness.* The algorithm should serve fairly among the input ports.
- *Starvation avoidance.* For high performance, the algorithms basically should be starvation-free, which means that a nonempty VOQ should never be remained without service indefinitely.
- *Implementation complexity.* High implementing complexity results in long scheduling time that limits the line speed. Thus, the algorithm must be easy to be implemented on the hardware [25].
- *Scalability.* The algorithm should be scalable to high line speeds and large number of ports. That is, the algorithm can be implementable when the line rates or the number of ports increase.
- *Existence of per-class QoS support.* The algorithm can supply different QoS support for different classes of flows. In that way, the different QoS requirements of each flow can be satisfied.
- *Variable packet size support.* The fabric scheduler can perform better for the metrics that are described above if it is aware of the fragmentation and reassembly of the IP packets into fixed size cells.

2.2.4 Selected Previous Work on Fabric Schedulers

There are mainly two different approaches to fabric scheduling: the slot-based and the frame-based scheduling [29]. The frame-based scheduling algorithms compute a sequence of F matching decisions in a frame length which equals to F time slots.

The computed matching decisions are used in F consecutive time slots. Moreover, the matching decisions can be re-used repeatedly until long-term traffic demands change [10]. On the other hand, the slot-based scheduling algorithms compute the matching decision in each time slot continuously.

For IQ switches, it is hard to provide QoS guarantees without any information about oncoming packets while dealing with the matching problem in each time slot [30]. In the frame based algorithms, incoming packets are stored in a buffer for a frame (F consecutive time slots); and therefore, provision of QoS guarantees are easier and gives more reliable results. Furthermore, for this matter, a flow based Iterative Packet Scheduling(FIPS), which is a frame based scheduling algorithm, is proposed in [30]. Additionally, a frame based scheduling algorithm that can limit the delay jitter and achieve 100% throughput while maintaining the unity switch speedup is proposed by Szymanski in [21]. Another frame based fabric scheduling algorithm the Iterative Scheduling with No Priority (ISNOP) is proposed in [29] to supply better QoS for differentiated classes. In contrast, the frame based scheduling algorithms are too complex to implement on hardware when they are compared with the slot based scheduling algorithms, Table 2.2.7. To the best of our knowledge, they are not preferred in commercial use up to date because of high implementation complexity.

Unlike frame based scheduling algorithms, slot based scheduling algorithms compute matching for each time slot independently. By doing so, they can adapt to dynamically variable traffic patterns. Yet, the time slot will be shorter because of the increase in line speed and this decrease in the time slot limits the computation time for slot based scheduling algorithms.

Hopcroft and Karp [31] proposed the fastest algorithm for Maximum Size Matching which is a slot-based scheduling algorithm. This algorithm guarantees the maximum matching possible and throughput. However, it cannot supply fairness due to starvation of flows. Also, even for suitable traffic patterns, the algorithm may be unstable [32]. The maximum weight matching (MWM), in contrast, works stably with admissible traffic patterns. The weight function can be related to either queue length or queuing delay of the head-of-line packet. The former one is called as Longest Queue First (LQF) and the latter one is called Oldest Cell First (OCF) [2]. On the other

hand, MWM is difficult to parallelize and it works in sequential manner. Thus, it has a time complexity of $O(N^3)$. In spite of the fact that these algorithms are both stable and capable of high throughput and low delay, they are not suitable for high-speed switches due to their implementation and time complexity. Indeed, there is not any commercial chipsets on which MWM is implemented [26].

In order to decrease such implementation and time complexity, maximal matching algorithms are suggested. On the other hand, the Parallel Iterative Matching (PIM) is one of the well known scheduling algorithms which uses distributed maximal matching algorithm [33]. In PIM, the input and output ports are matched randomly in an iterative way, and therefore, it is not a fair scheduling algorithm. The fairness problem is eliminated by iSLIP [7] algorithm. Here, the randomness is changed with a round-robin fashion by input and output ports. The most of dynamic schedulers use sub-optimal heuristic schedulers which can achieve high throughput, delay and jitter performance. Moreover, Cisco 1200 series routers developed the heuristic iSLIP algorithm which supply about 100% throughput for uniform traffics [10]. Moreover, Dual Round-Robin (DRR) is proposed in [34]. In the DRR, each input arbiter performs request selection;therefore, the accept state in iSLIP is not required. Thus, DRR requires less time to perform arbitration.

There are also studies on heuristic algorithms in order to better approximate to MWM. The iterative Longest Queue First (iLQF)[35] algorithm which supplies better approximation to MWM than iSLIP. iSLIP does not care about queue lengths but iLQF does. iLQF can be designed with same architecture with iSLIP but iLQF requires to change larger amount of information between arbiters [26]. Moreover, WP-iLQF algorithm which concentrates on exchange messages rather than queue length is proposed in [26]. WP-iLQF has time complexity of $O(N)$. Depending on the simulation results [26], it is claimed that WP-iLQF supplies lower packet delay then iLQF. Additional to these algorithms, a heuristic scheduling policy, named as deadline-aware maximum weight matching (MWM), for multimedia streaming applications is proposed in [36] in order to minimize the packet loss ratio in the input queues. On the other hand, these algorithms have more difficult to implement in hardware when they are compared with maximal matching algorithms such as PIM, DRR and iSLIP.

2.2.5 ISLIP

iSLIP is a widely used fabric scheduling algorithm that achieves high throughput and that can easily be implemented in the hardware. The 50-Gb/s IP router Cisco 12000 GSR and Tiny Tera(a 0.5-Tb/s MPLS switch) are developed on iSLIP algorithm [7]. The iSLIP uses round robin arbiters at each input and output ports. Thus, the fairness problem in PIM is overcome by iSLIP. The iSLIP algorithm consists of three steps for each iteration. All unmatched input and output pairs are involved in the next iteration. Moreover, the round robin arbiter pointers are updated only in the first iteration. In this way, starvation is also avoided.

Step1: Request. Each input sends a request to outputs for which it has a packet [7].

Step2: Grant. If an output receives any requests, the one which is next in the round robin grant arbiter is chosen, starting from the highest priority one. Input port is informed, if its request is granted. In the first iteration only, the round robin grant arbiter pointer is incremented by modulo N to one location beyond the granted input, if and only if, the grant is accepted in step 3 of the first iteration [7].

Step3: Accept. If an input is granted by multiple output ports, it accepts the one in the same manner in step 2. In the first iteration only, the round robin accept arbiter pointer is incremented (modulo N) to one location beyond the accepted output [7].

With the help of round robin schedulers, iSLIP serves fairly and no connection is starved. In the iSLIP, a new matching graph should be found within a time-slot [10]. It is claimed that, for high link rates, the duration of the time slot will decrease and that causes very limited execution time for iSLIP. When the link rates increase, the clock frequencies in the hardware design also increase. The higher clock frequency means faster decisions. Thus, that will not be a problem for iSLIP implementation.

An example of the iSLIP steps for the first iteration can be found in Fig.2.2, Fig.2.3 and Fig.2.4.

The input port 1 has one packet destined to the output port 1 and 4 packets which are destined to the output port 2. There is not any packets in the input port 2, whereas there are 2 packets in the input port 3 which are destined to the output port 2 and

output port 4. Lastly, the input port 4 has 3 packets whose destination is the output port 4. The input ports, which have queued packets, send requests to output ports as seen in Fig. 2.2.

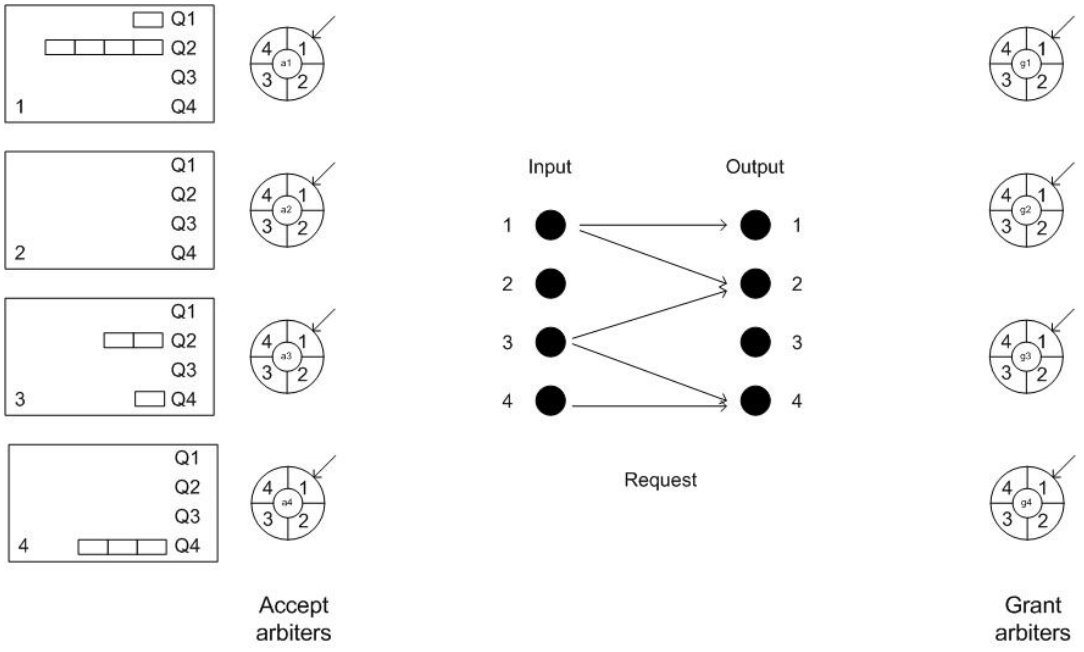

Figure 2.2: Request step (step1) of an iSLIP iteration

If an output port receives only one request, that request is granted. In the case that an output port receives multiple requests, the grant is decided by checking its grant arbiter in a round-robin manner. In our example, the complete grant signals can be seen in Fig.2.3. The output port 1 has only one request from the input port 1. Thus, the output port 1 sends grant to the input port 1. The output port 2 has multiple requests from both the input port 1 and the input port 3. The grant arbiter pointer of the output port 2 points to the input port 1. In this way, the output port 2 chooses the input port 1 and sends grant signal to it. There are multiple requests for the output port 4 from the input port 3 and input port 4. The grant arbiter pointer of the output port 4 points to the input port 1, neither to the output port 3 and nor to the output port 4. In a round-robin manner, the closest one which is the input port 3 is selected in that condition.

If an input port receives only one grant, that grant is accepted. However, if an input port receives multiple grants, the closest one in round-robin manner is accepted. In our case, the accept signals can be seen in Fig.2.4. The input port 1 receives multiple

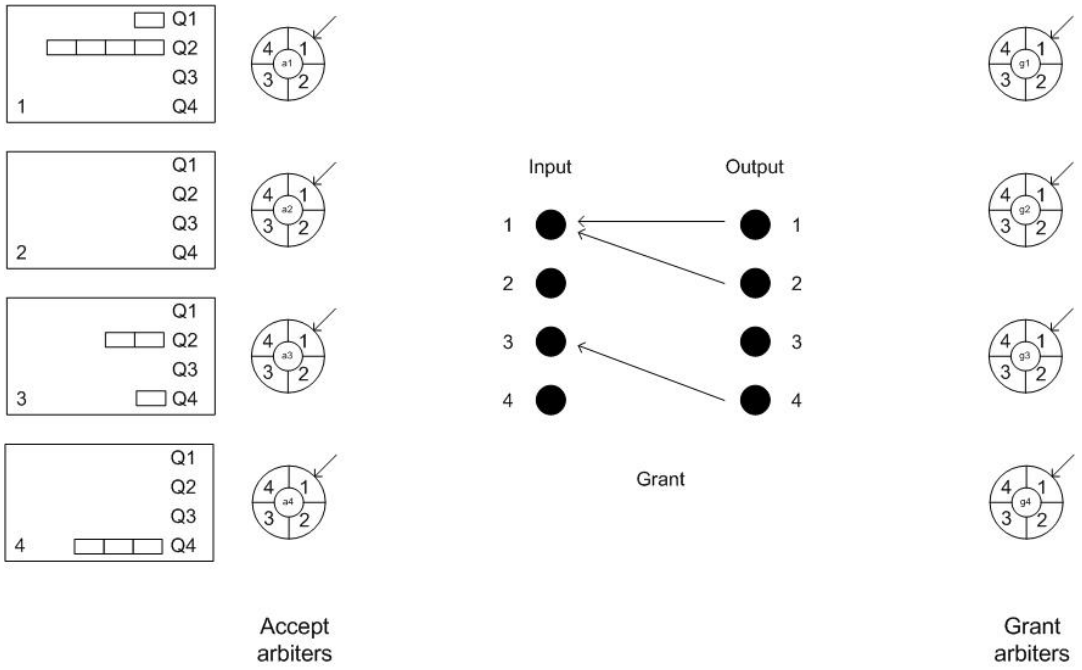

Figure 2.3: Grant step (step2) of an iSLIP iteration

grants from output 1 and output 2. The output port 1 is accepted since the accept arbiter pointer of the input port 1 points to the output port 1. The input port 3 receives single grant from the output port 4. Thus, the input port 3 accepts the grant of the output port 4.

In the first iteration, after the 3 steps, the accept arbiter pointers and grant arbiter pointers which belong the matched input ports and output ports must be updated. If the grant of an output port is accepted, the grant arbiter pointers of that output port must also be updated. Moreover, the accept arbiter pointer of the input port which sends accept to any output port must be also updated. In our example, the grant of the output port 1 is accepted by the input port 1. Hence, the grant arbiter pointer of the output port 1 is shifted to position 2. The accept arbiter pointer of input port 1 is also shifted to position 2 since the input port 1 accepts the output port 1. Furthermore, the grant arbiter of the output port 4 is shifted to position 4 since the input port 3 accepts the grant of the output port 4. In the same manner, the accept arbiter of the input port 3 is shifted to position 1 because the grant of the output port 4 is accepted by the input port 3. The pointers' update is only processed in the first iteration as mentioned above. In the next iterations, the pointers are not updated.

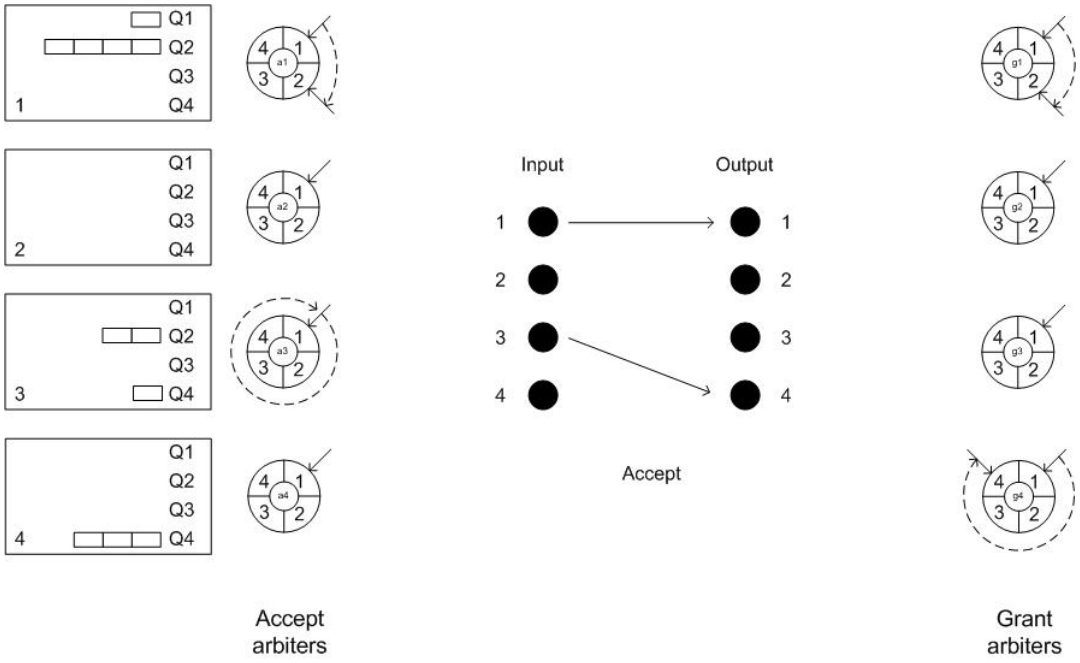

Figure 2.4: Accept step (step3) of an iSLIP iteration

To sum up, the features of iSLIP are:

- VOQ organization at the input ports
- Three step messaging as request grant and accept to decide the input output matching
- Round robin arbitration in selecting the sent requests and grants at the inputs and outputs respectively.

There are 3 variations of iSLIP such as prioritized iSLIP, threshold iSLIP and weighted iSLIP [7].

The prioritized iSLIP and the weighted iSLIP give different static priorities to the flows to achieve QoS. The threshold iSLIP, on the other hand, employs dynamic priorities in the prioritized iSLIP. These dynamic priorities for the queues are computed according to the amount of cells in the queue to maximize throughput.

2.2.5.1 Prioritized ISLIP

In the prioritized iSLIP, there are separate FIFO for each output and priority level. For an NXN router with P priority levels, there are PxN FIFO's in each input. The queue between an input i and an output j at priority level l is labeled as $Q_l(i, j)$, where $1 \leq i, j \leq N, 1 \leq l \leq P$.

The iSLIP algorithm is changed as follows.

Step1: Request. In the input i, the highest priority nonempty queue for output j is chosen. Then, the priority level l_{ij} of this queue is sent to output j.

Step2: Grant. If output j receives multiple requests, firstly it selects the highest level requests, $L(j) = max_l(l_{ij})$. Then, the output chooses one input among those requests by using output grant arbiter. The output grant arbiter works with separate pointer g_{jl} for each priority level. The grant arbiter pointer $g_{jL(j)}$ is used during the selection among the requests at level $L(j)$. The same round-robin scheme is used with basic iSLIP. The inputs are informed whether their requests are granted or not. The pointer $g_{jL(j)}$ is incremented in the same way as basic iSLIP, if and only if, the input i accepts the output j in the step 3 of the first iteration.

Step3: Accept. If an input receives any grants, the highest level grant is searched, $L'(i) = max_l(l_{ij})$. Then, an output among those grants at level $l_{ij} = L'(i)$ is chosen by using input accept arbiters. A separate pointer a_{il} is kept for each priority level. The accept arbiter pointer $a_{iL'(i)}$ is used during the selection among the grants at level $L'(i)$. The selection is made in a round-robin manner. The input informs each output about whether its grant is accepted or not. Then, the accept arbiter pointer $a_{iL'(i)}$ is incremented (modulo N) to one location beyond the accepted output.

The hardware implementation of the prioritized iSLIP is more complex than the basic iSLIP since separate pointers are maintained for different priority levels.

2.2.5.2 Threshold ISLIP

In the threshold iSLIP, the maximum-sized match and maximum-weight match are compromised. The fullness of the queues are quantized according to a set of threshold values. Afterwards, the priority levels in the priority iSLIP algorithm are calculated by using those threshold values. There is an ordered set of threshold values for each input queue, $T = \{t_1, t_2, \ldots, t_T\}$, where $t_1 < t_2 < \ldots < t_T$. Input arranges the priority level of $l_i = a$ when $t_a \leq Q(i, j) < t_{a+1}$.

2.2.5.3 Weighted ISLIP

The strict priority scheme of prioritized iSLIP may not be suitable for many conditions since prioritized iSLIP results in starvation of low prioritized queues. In such cases, throughput to an output can be divided among the inputs according to weights of inputs.

In the basic iSLIP, an ordered circular list $S = \{1, \ldots, N\}$ is used by arbiters. In the weighted iSLIP algorithm, the ordered list at the output j is used as $s_j = 1, \ldots, W_j$, where $W_j = lowestcommonmultiple(d_{i,j})$ and the input i appears $(n_{ij}/d_{ij})xW_j$ times in S_j.

2.2.6 Variable-size Packet Switching

The switch architectures are able to deal with variable-size packets by working on fixed-size cells internally. Those variable-size packets are segmented into the fixed-size cells while entering the switch architecture. Then, scheduling and switching operations are realized on the fixed-size cells. While leaving the switch architecture, the fixed-size cells belonging to the same packet are reassembled at output ports.

The scheduling algorithms can be divided into two groups as packet-mode scheduling and cell-mode scheduling. In the packet-mode scheduling algorithms, all fixed-size cells belonging to the same packet are transferred to output ports contiguously by the scheduling algorithm. In that way, it is easy to reassemble at output ports since the fixed-size cells belonging to the same packet are transferred to output ports with-

out being interrupted by the other fixed-size cells. That condition saves memory and reduces complexity in the reassembling circuit at output ports. The rest of the scheduling algorithms, which do not care about contiguous transfer of the fixed-size cells belonging to the same packet, are called as cell-mode scheduling algorithms.

The switch architectures are categorized into two: cell switches and packet switches [28]. On the one hand, the cell switches work on fixed-size cells. On the other hand, the packet switches are designed to be based on the cell switches in order to handle with variable-size packets.

In the architecture, there is the Input Segmentation Module(ISM) at each input port. The variable-size packets are segmented into the fixed-size cells by the ISM. The packets enter into the ISM on the packet line speed (PLS). The ISMs work in the store-and-forward mode. The segmentation process starts after the reception of the whole packet. The fixed-size cells which are created after segmentation are transferred to cell-switch at the internal line speed (ILS). The PLS is incremented to the ILS in order to account for segmentation overheads. Then, all cell-switch operations are proceed at the ILS. The switching fabric transfers the fixed-size cells to output reassemble module (ORM) at the ILS. If packet-mode scheduling algorithms are not used as fabric scheduler, the fixed-sized cells belonging to different packets can be successively transferred to the ORM. In this way, there are different queues for each input port in ORM. On the other hand, at most one packet can be completed in each time slot since at most one fixed-size cell can reach ORM in each time slot. When a packet is completed in the ORM, it is transferred to the packet FIFO. Next, the variable-size packets leave the switch at line speed since segmentation overheads are discarded by the ORM.

If packet-mode scheduling algorithms are used, the ORM is not necessary anymore. In that way, the packet switch architecture can be simplified. However, here it can be seen that the long packets unfairly block the short packets.

2.2.7 Fabric Schedulers Supporting per Class QoS

iSLIP [7] is a slot-based scheduling algorithm which makes matching decision per slot basis. The iSLIP works under unity speedup condition. Furthermore, it has a time complexity of $O(1)$ and implementation complexity of $O(N^2)$. The hardware implementation is explained in [7]. In the iSLIP implementation, none of input ports suffer from starvation, and yet it does not supply different QoS per class.

In order to supply differentiated QoS per class, the prioritized iSLIP is proposed in [7]. It is also a slot-based scheduling algorithm and it does not require speedup more than 1. In the prioritized iSLIP, different VOQs are implemented for different classes. Its complexity is more than basic iSLIP; however, the input ports which have lower priority suffer from starvation since higher priority levels always have right to send packets if they have any packets.

Table 2.1: Comparision of the existing algorithms.

	class	throughput	time complexity	QoS support per class	implementation
iSLIP[7]	slot-based	no starvation	$O(1)$	no per class	hardware
prioritized iSLIP[7]	slot-based	starvation for low priorities	not defined	per class	hardware
threshold iSLIP[7]	slot-based	no starvation	not defined	per class	hardware
weighted iSLIP[7]	slot-based	no starvation	$O(1)$	no per class	hardware
SRA+[37]	slot-based	no starvation	$O(1)$	per class	simulation
RFSMD[21]	frame-based	no starvation	$O(NFlogNF)$	per class	simulation
ISNOP[29]	frame-based	no starvation	$O(kn^{4.5} + k^2n^3)$	per class	simulation
FIPS[30]	frame-based	no starvation	$O(kn^{4.5})$	per class	simulation
2-LLP IMC VOQ iSLIP Chapter4	slot-based	no starvation	$O(1)$	per class	FPGA implementation

In the threshold iSLIP [7], the priority levels for input ports are dynamically arranged

according to the occupancy of queues. In that way, starvation problem in the prioritized iSLIP is tried to be solved. On the other hand, the threshold iSLIP cannot be fair in terms of classes. If a class sends more packets, the occupancy of the queues for that flow are always high. Thus, the threshold iSLIP gives priority for the flow which sends more packets than others.

In some cases, the strict priority is undesirable. When it is the case, the weighted iSLIP is claimed to be preferable in [7]. In the weighted iSLIP, there are no different VOQs for different classes. In the arbiters, some input ports have more chance. To this end, the weighted iSLIP does not supply differentiated QoS per class, but rather it serves input ports in a weighted manner.

Single round robin(SRA) scheduling is proposed in [37]. The SRA supplies different bandwidth guarantees for different classes. Moreover, it has time complexity of $O(1)$. Therefore, it is easy to implement on the hardware. Additionally, it does not concern about delay guarantees for classes. That is, the delay of the flows are uncontrolled. Considering all, the SRA supplies differentiated QoS for classes in terms of bandwidth only.

Similarly, a SRA+ is proposed in [37] in order to limit the delay of the classes while supplying bandwidth guarantees for classes. The time complexity of SRA+ is also $O(1)$. However, SRA+ does not differentiate QoS in terms of packet delay. In the SRA+, all classes have the same delay while they are serviced according to their bandwidth requirements. Hence, it can be said that the difference in delays of different classes is eliminated in both SRA and SRA+.

According to Szymanski [21], the frame-based Recursive Fair Stochastic Matrix Decomposition (RFSMD) is suitable, since it works under unity speedup. It stores the packets for a frame which equals to F time-slot period. Then, it gives a decision for F time-slot. The differentiated delay guarantees can be acquired in that scheduling algorithm. On the other hand, one frame delay is naturally added for output packets since packets are stored for a frame length in order to start decision process. Moreover, its time complexity is $O(NFlogNF)$. In order to handle with this time complexity, the frame length can be increased but it is limited by hardware sources. The memory space for storing packets for frame length must be raised to also increase the frame

length. The hardware implementation is not explained in [21] to clarify whether it can be possible to implement on hardware or not. Only simulation results are discussed.

In the article written by Lee [30], Flow based Iterative Packet Scheduling (FIPS) is proposed in order to supply differentiated QoS for different classes in terms of delay guarantees. The FIPS, as a frame-based scheduling algorithm, has a high success rate and a low packet dropping ratio. It works under unity speedup condition but also it has a time complexity of $O(kn^{4.5})$. In the article [30], only the simulation results are presented, but there is nothing about its hardware implementation. Moreover, it is claimed that the FIPS's performance is not enough to be used in practice even though it has higher throughput than the Earliest Deadline First (EDF) based algorithms [29].

The Iterative Scheduling with No Priority (ISNOP) is proposed in order to solve multi class deadline guaranteed packet scheduling problem [29]. The ISNOP is a frame-based scheduling algorithm which does not require speedup more than unity. In [29], it is claimed that the ISNOP has more success rate then the FIPS for high loads. For high switch sizes, the ISNOP always gives more success rate than the FIPS. In [29], it is also claimed that, the ISNOP supplies lower packet dropping ratio than all previous frame-based algorithms. Still, it is debatable that it can be implemented on hardware or not since it has a time complexity of $O(kn^{4.5} + k^2n^3)$.

In this thesis, a 2-level limited prioritized(LLP) IMC VOQ iSLIP is proposed which is a slot-based scheduling algorithm. Its time complexity equals to $O(1)$ and it works under unity speedup condition. Thus, its hardware implementation, as it is explained in chapter 4, is quite easy. Not only it behaves fair for all classes, but it also does not suffer from starvation. It supplies differentiated QoS for 2 different classes in terms of packet delay.

CHAPTER 3

IMC VOQS: INTELLIGENT MULTI CLASS VOQS

3.1 IMC VOQ Overview

Differentiating traffic classes at the network layer in the switches and routers is a desired feature to support end to end QoS.

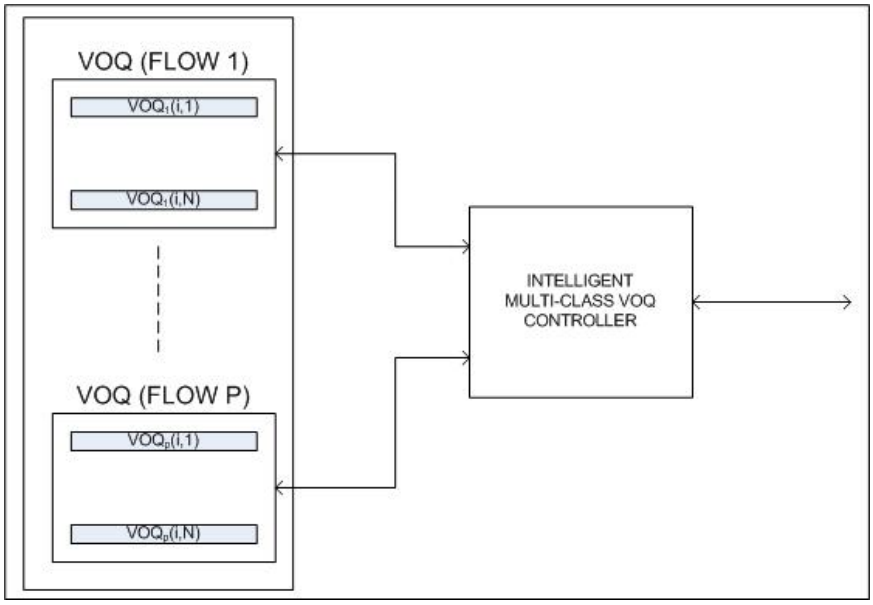

Figure 3.1: The proposed intelligent multi-class VOQ architecture at input port i

Consider an NxN fabric that supports P different priority levels with PxN VOQs that work as FIFO in each input port as shown in Fig. 3.1. The queues at input i for output j are labeled as $Q_l(i, j)$; where l labels the priority level, $1 \leq i, j \leq N, 1 \leq l \leq P$ and the highest priority is $l = P$. Hence, there are up to P packets that are ready at input i for output j. However, the well-known and tested fabric scheduler algorithms such

as iSLIP[7, 38] are designed for VOQs arrangements where each input i sends only a single request to output j. The Intelligent Multi-class (IMC) unit that is proposed in this thesis is a preprocessing unit as shown in Fig. 3.1 that generates this single request out of P queues at input i for a given output j. Hence, this makes it possible for a given fabric scheduler to support different classes without changing its operation to be different than the prioritized iSLIP and Weighted iSLIP in [2] that add multiple arbiters or change the standard round robin iSLIP arbiters.

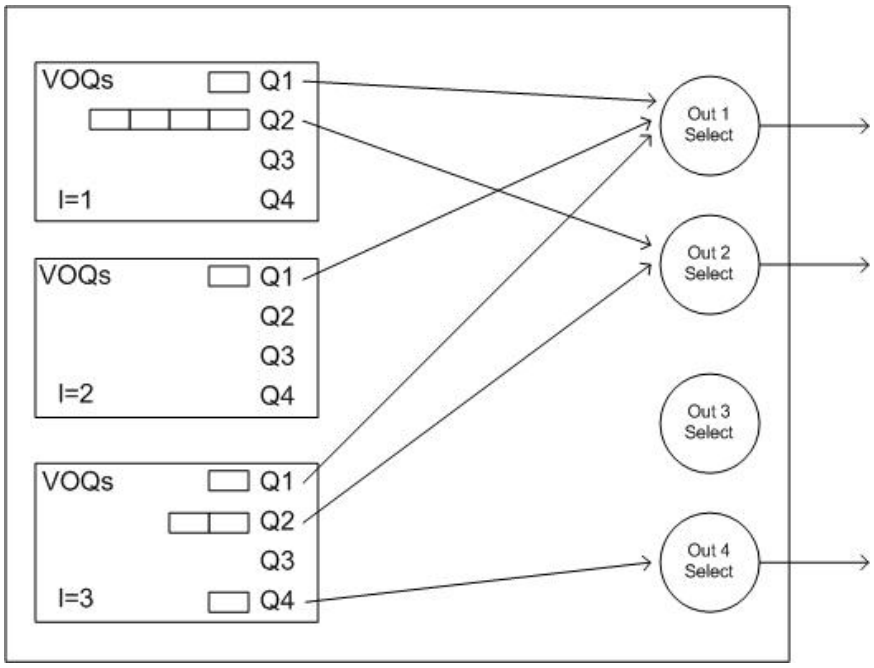

Figure 3.2: an example of IMC VOQ at input port i

In Fig.3.2, VOQs blocks for each priority level are represented in input port i. That example belongs to a 4x4 switch supporting 3 different priority levels. In the example, priority level 1, priority level 2 and priority level 3 have cells for the output port 1. Moreover, there are 2 cells for the output port 2. One of these two is from priority level 1 and the other one is from priority level 3. There is not any cells for the output port 3. Lastly, there is one cell for the output port 4. Considering all, there are 3 candidate requests for the output port 1, 2 candidate requests for the output port 2 and 1 candidate request for the output port 4. The intelligent controller block must decide *atmost* one request to each output port for the given input port i. Although there is a packet which is destined to an output port, the request may not be sent to that output port depending on the decision algorithm.

On the other hand, when matching decision is given by fabric scheduler, the intelligent controller block is responsible for reading the VOQs. For instance, in Fig.3.2 the output port 1 and the input port *i* are matched by fabric scheduler. After the scheduling is completed, the input port *i* is informed about that the output port 1 waits for the packet. In that time, there are 3 available packets for the output port 1 belonging to level 1, level 2 and level 3. Thus, the intelligent controller block should memorize that the requests which are lastly sent to output ports belong to which priority level.

3.2 IMC VOQ Decision Algorithms

In an input port, the selection among candidate requests for an output port can be done in many ways by IMC VOQ unit. 2 different approaches can be followed here. The first one is the *separated* selection for each output port. In this case, all requests to output ports can be decided independently from each other. On the other hand, the second approach is the *combined* selection, in which all requests to output ports must be from the same priority level. In this second approach, it is enough to memorize only 1 priority level which wins the competition in order to read right queues. Whereas, in the first approach, priority levels for each request sent to output ports can be different; and therefore, N priority levels must be memorized which causes an implementation complexity of $O(N)$. Although the throughput for some traffic scenarios can be low for the combined selection algorithms, their implementation complexity is lower .

It is possible to implement various decision algorithms for request selection. Furthermore, 2 different algorithms are proposed as an example in this thesis. The first one is a 2-level strictly prioritized(2-LSP) IMC VOQ algorithm. The second one is a 2-level limited prioritized(2-LLP) IMC VOQ algorithm. Both of the 2 algorithms support 2 priority levels. Moreover, both algorithms are based on the combined selection approach in order to keep the implementation complexity low.

3.2.1 The 2-Level Strictly Prioritized IMC VOQ Algorithm

There are 2 priority levels in the 2-level strictly prioritized (2-LSP) IMC VOQ algorithm. The algorithm is based on the combined selection approach; therefore, the candidate requests belonging to decided priority level are sent to output ports.

In 2-LSP IMC VOQ algorithm, if there is a packet in any of the queues belonging to a higher priority level, the queues in the higher priority level always have right to send request. If all queues in the higher priority level are empty, the queues in lower priority level have right to send requests. That algorithm results in starvation for queues in lower priority level.

The pseudo code of 2-LSP IMC VOQ algorithm can be found in Algorithm 1. The algorithm runs for each input port in parallel way and independently. The pseudo code represents the code running in input port i.

Algorithm 1 The 2-Level Strictly Prioritized IMC VOQ Algorithm for input i

```
1:  int LastChosenPriorityLevel = 0
2:  bool FinalRequest[N]        ▷ FinalRequest(j) represents the decided request from
    given input i to output j
3:  bool CandidateRequestVOQ1[N]    ▷ RequestVOQ1(j) represents the candidate
    requests for priority level l=1
4:  bool CandidateRequestVOQ2[N]    ▷ RequestVOQ2(j) represents the candidate
    requests for priority level l=2
5:  while 1 do
6:      for j ≤ N do                    ▷ checks Q2 whether it has any packet or not
7:          if Q2(i, j) is non − empty then
8:              flag2 = 1
9:              break
10:         end if
11:     end for
```

```
12:     if flag2 then
13:         FinalRequest(:) = CandidateRequestVOQ2(:)
14:         LastChosenPriorityLevel = 2
15:     else
16:         FinalRequest(:) = CandidateRequestVOQ1(:)
17:         LastChosenPriorityLevel = 1
18:     end if      ▷ Request selection is done. Wait for the fabric scheduler decision.
19:     if LastChosenPriorityLevel == 1 then
20:         READ(VOQ1)
21:     else
22:         READ(VOQ2)
23:     end if
24: end while
```

3.2.2 The 2-Level Limited Prioritized IMC VOQ Algorithm

In order to avoid the starvation problem, a 2-level limited prioritized (2-LLP) IMC VOQ algorithm is proposed. If the queues in a higher priority level have strict priority like 2-LSP IMC VOQ algorithm, the queues in the low priority level suffer from the starvation problem. Thus, a limit value is defined in 2-level limited prioritized (2-LLP) IMC VOQ algorithm. That value restricts the number of requests which can be sent by the queues in the high priority level while queues in the low priority level are non-empty. When the limit value is reached, the queues in the high priority level must wait for the low priority queues' sending the request. The algorithm does not care about whether the requests are granted by the fabric scheduler or not. Depending on that limit value, queues in the lower priority level may be served even if the queues in the higher priority level are not empty. In this way, the starvation is avoided.

The pseudo code which is written for 2-LLP IMC VOQ algorithm can be found in Algorithm2. It runs in a parallel way for each input port independently.

Algorithm 2 The 2-Level Limited Prioritized IMC VOQ Algorithm in input port *i*

```
1: int LimitCounter = 0
2: int LimitValue = MaxSuccesiveAllowedRequests
```

```
int LastChosenPriorityLevel = 0
bool FinalRequest[N]        ▷ FinalRequest(j) represents the decided request from given input i to output j
bool CandidateRequestVOQ1[N]    ▷ RequestVOQ1(j) represents the candidate requests for priority level l=1
bool CandidateRequestVOQ2[N]    ▷ RequestVOQ2(j) represents the candidate requests for priority level l=2
while 1 do
    for j ≤ N do                ▷ checks Q1 whether it has any packet or not
        if Q1(i, j) is non − empty then
            flag1 = 1
            break
        end if
    end for
    for j ≤ N do                ▷ checks Q2 whether it has any packet or not
        if Q2(i, j) is non − empty then
            flag2 = 1
            break
        end if
    end for
    if flag1&flag2 then
        if LimitCounter == LimitValue then
            FinalRequest(:) = CandidateRequestVOQ1(:)
            LastChosenPriorityLevel = 1
            LimitCounter = 0
        else
            FinalRequest(:) = CandidateRequestVOQ2(:)
            LastChosenPriorityLevel = 2
            LimitCounter = 0
        end if
    else
```

```
31:         if flag1 then
32:             FinalRequest(:) = CandidateRequestVOQ1(:)
33:             LastChosenPriorityLevel = 1
34:         else
35:             FinalRequest(:) = CandidateRequestVOQ2(:)
36:             LastChosenPriorityLevel = 2
37:         end if
38:     end if
        ▷ Request selection part is done. ▷ Waiting for the fabric scheduler decision.
39:     if LastChosenPriorityLevel == 1 then
40:         READ(VOQ1)
41:     else
42:         READ(VOQ2)
43:     end if
44: end while
```

In Fig.3.3, how the 2-LLP IMC VOQ algorithm works is explained by using an example. In the example, content of VOQs belonging to an input port are presented for 4 successive time slots. In that example the limit value is 3, in other words, 3 successive requests are allowed for queues in the high priority level when any of the queues in the low priority level has at least one packet. Moreover, it is assumed that all requests are accepted by output ports in each time-slot. Thus, the cells which sent request are transferred to the switch fabric.

At the time slot 1, the limit counter equals to 0. In this way, the candidate requests belonging to the high priority level are sent. Then, the limit counter is incremented to 1 since there is at least one packet in the low priority queues. In the second time slot, the limit counter equals to 1. Here, the high priority level queues still have priority. After, the limit counter is incremented to 2. At the time slot 3, the higher priority level queues send request again since the limit counter equals to 2. Then, the limit counter is incremented to 3 which equals to the limit value in the example. At the time slot 4, finally, the limit counter reaches to the limit value. Thus, the requests belonging to the low priority queue are sent. Afterwards, the limit counter is reset to 0 at the end of the time slot 4.

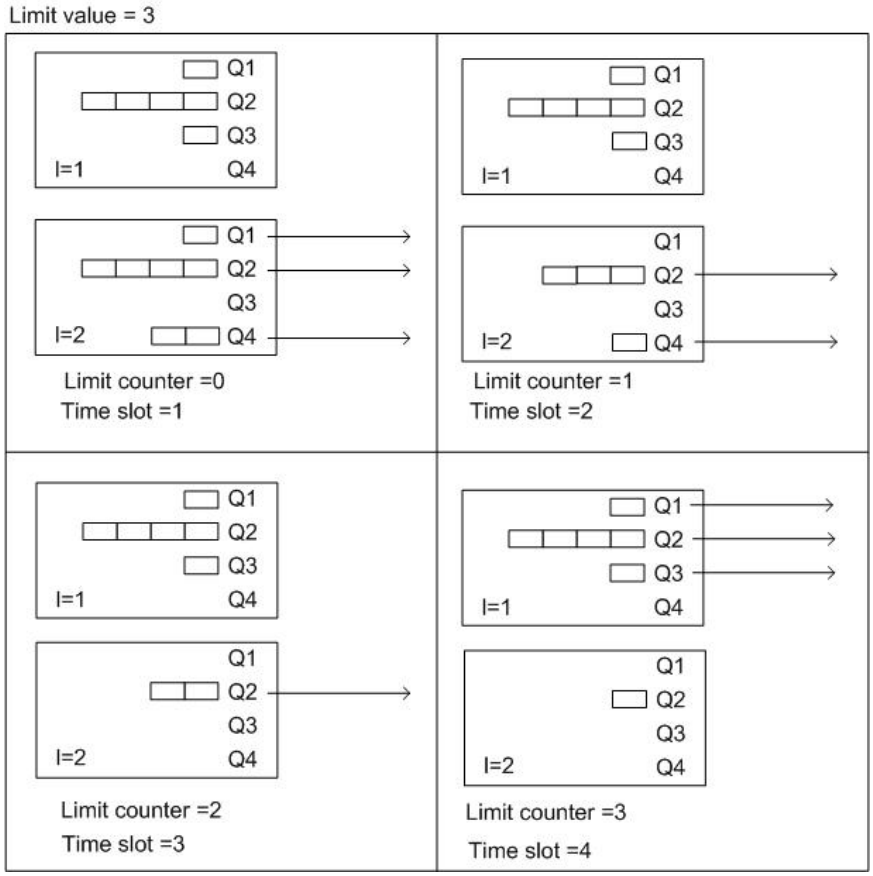

Figure 3.3: An example of 2-level limited prioritized IMC VOQ algorithm

CHAPTER 4

FPGA IMPLEMENTATION OF $4X4$ 2-LLP IMC VOQ ISLIP ROUTER

The proposed IMC unit is combined with a fabric scheduler in order to provide low packet delay for high priority level packets.

iSLIP is widely used fabric scheduling algorithm since it is easy to scale and easy to implement. Moreover, with the help of round robin arbitration output lines are fairly served and the starvation of any non-empty queue is avoided. Thus, iSLIP is preferred in order to implement on FPGA as a fabric scheduler. The proposed IMC VOQ structure which is independent from fabric scheduling algorithm is combined with iSLIP. IMC VOQ structure can be combined with any other algorithms such as DRR and PIM.

There are 2 proposed algorithms for IMC VOQ in chapter 3. The 2-LSP IMC VOQ results in starvation;therefore, 2-level limited prioritized(2-LLP) IMC VOQ algorithm is developed. To this end, 2-LLP IMC VOQ algorithm is chosen to implement instead of 2-LSP IMC VOQ algorithm. In this way, 2 priority levels are supported in the implemented $4x4$ router. Furthermore, supporting at least 2 priority levels is enough in order to implement the QoS differentiation in practice.

Furthermore, the overall architecture supports variable-size packets. In order to support variable-size packets, the received packets are segmented into fixed-size cells similar to Cisco Cells [6] before they are stored in VOQs. The fabric scheduling algorithm works on fixed-size cells. After scheduling is completed, fixed-size cells are transferred to output line blocks. At the output line blocks, when all segments of

the packets are completed, the fixed-size cells are reassembled back into variable-size packets and delivered to output line.

The design of the overall architecture is implemented on FPGA by using VHDL hardware design language. FPGAs are suitable for the implementation of high-speed design architectures because of their parallel operating feature with reconfigurable and adaptive processing capability. With the help of parallel operating ability, all incoming packets from different input ports can be processed at the same time in FPGA. Moreover, high-speed designs can be implemented on FPGAs. While considering 10Gbps or even higher network speeds, FPGAs are most suitable development platforms for switches/routers.

4.1 General View of FPGA Implentation

The general view of FPGA implementation of 4x4 switch fabric can be seen in Fig. 4.1. There are 4 identical input line blocks and 4 identical output blocks. There are a switch fabric module and a fabric scheduler which work cooperatively in order to transfer packets from input line blocks to output line blocks. There is a time slot trigger block which is responsible for synchronizing all the blocks by measuring time-slot period.

The fabric scheduler block works on fixed-size cell. Similar to cisco cells, the fixed-size cell length is chosen as 64-byte, 4-byte is header and the rest is payload. Switch fabric and other blocks are designed in order to work on 32-bits data in one clock. Thus, transferring full packet (64x8 bits) through switch fabric takes 16 clocks. That is, time-slot period equals to the fixed-size cell time which equals to 16 clocks.

While designing iSLIP, the hardware structure in [7] is taken as a reference. Thus, all input ports are directly connected to fabric scheduler block in order to submit requests. Moreover, the decision of the fabric scheduler is directly submitted to input ports and switch fabric as shown in Fig.4.1. Moreover, 2 iterations are done by iSLIP fabric scheduler block in order to increase throughput.

In order to transfer cells from input ports to output ports a 4x4 crossbar switch fabric

is designed. While designing switch fabric muxed-based crossbar architecture which is discussed in [39] is adopted.

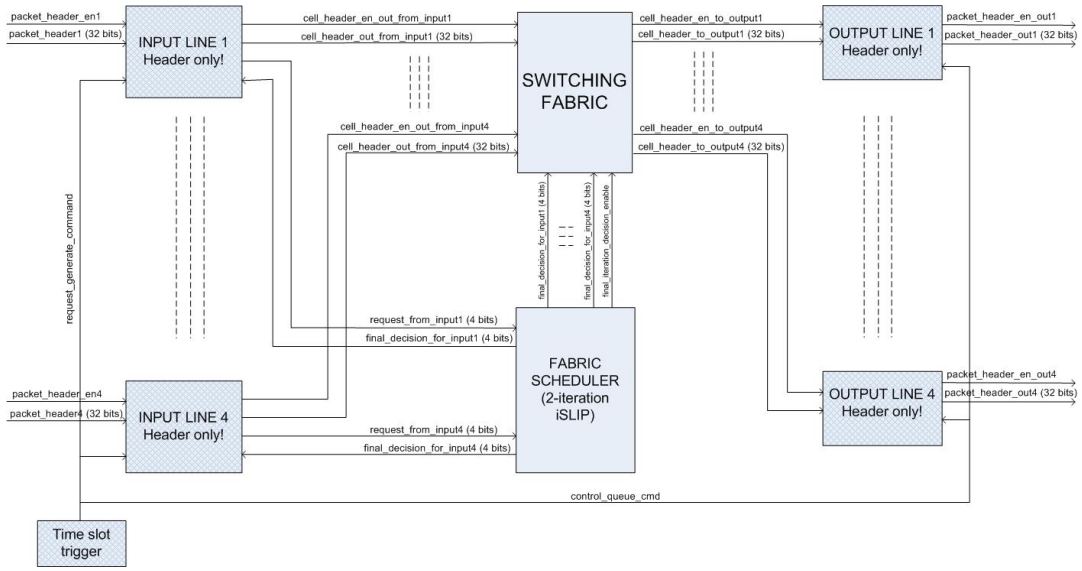

Figure 4.1: General view of FPGA implementation

As it is explained in [28], after segmentation size of the packet is larger than the original one because of the new produced cell headers. Note that, in order to deal with segmentation overheads, the fabric should have speedup to forward the fixed-size packets at line speed. In this thesis we propose that cell headers are stored inside the FPGA and cell payloads are assumed to be written to RAMs outside the FPGA and hence the fabric and the scheduler work at line speed. When the payloads are separated from headers, there is no need to increase speed of hardware. On the other hand, another switch fabric is required to transfer the payload.

In the Fig.4.2, the example about transferring the fixed-size cells after segmentation through 2 paths and 1 path can be found. In the first one, the cell payloads and cell headers are transferred through 2 different paths, one for cell payloads and the other for cell headers. In that one, there is no need to increase the hardware speed as shown in Fig.4.2. On the other hand, when the cell payloads and cell headers are transferred through one single line, the hardware must work faster in order to handle with segmentation overheads..

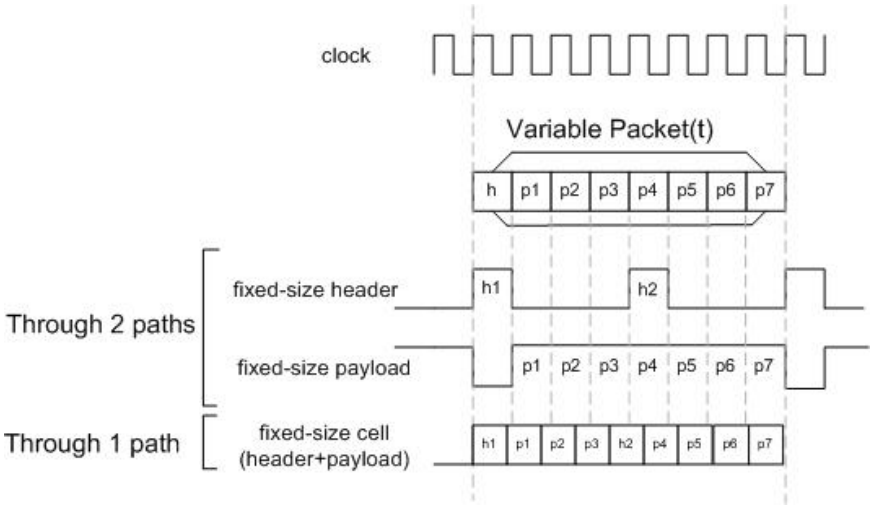

Figure 4.2: Transferring the fixed-size cells after segmentation

4.1.1 Variable Size Packet Header Format

The variable size packet header format which is used in FPGA implementation can be seen in Table. 4.1.

Table 4.1: Variable size packet header format

	Bit No	Fields
1 bit	31	Reserved
1 bit	30	Packet's class
2 bits	29-28	Packet's destination address
1 bit	27	Reserved
3 bits	26-24	Packet's size
2 bits	23-22	Reserved
2 bits	21-20	Packet's input address
20 bits	19-0	Packet's entering time

- Packet's class: it shows the type of class. In our implementation, there are two types of classes which are high priority class and low priority class. If packet belongs to high priority class, this bit will be 1. Otherwise, it will be 0.
- Packet's destination address: it defines the destination port of the packet. It is a 2-bits word. 4 different output ports can be defined. "00" represents output port 1, "01" represents output port 2, "10" represents output port 3 and "11" represents output port 4. When number of output ports are higher than 4, the

reserved bits can be used.

- Packet's size: it defines the packet size in terms of length of fixed-size cell. If it is "000", that means it's packet size is 1. If it is "111", it's packet size is 8. In that way, from 1 to 8 packet sizes can be defined.

- Packet's input address: it defines where the packet comes from. It is a 2-bits word like packet's destination address.

- Packet's entering time: it shows the time slot value when the packet enters into the router. At the entrance of the packet, time slot value of router is written to this area. Then, this area is used to measure the time between the packet's entering into the router and exiting from the router while packet is leaving from the router. In this way, the packet delay can be calculated in terms of time slot.

4.1.2 Fixed-size Cell Header Format

After segmentation of the variable size packets, new fixed-size cell headers are generated. The fixed-size packet header format which is used in FPGA implementation can be seen in Table. 4.2.

Table 4.2: Fixed-size cell header format

	Bit No	Fields
1 Bit	31	Reserved
1 Bit	30	Cell's Class
2 Bits	29-28	Cell's Destination Address
1 Bit	27	End Of Packet (EoP)
3 Bits	26-24	Packet's Size
2 Bits	23-22	Reserved
2 Bits	21-20	Cell's Input Address
20 Bits	19-0	Packet's Entering Time

- Cell's class: it contains same information with the packet's class field in variable size packet header. While producing new fixed-size cell header, that field is copied from the variable size packet header.

- Cell's destination address: it contains same information with the packet's destination address field in variable size packet header. While producing new fixed-size cell header, that field is copied from the variable size packet header.

- EoP: it defines whether the related fixed-size cell is the last fixed-size cell after segmentation or not for a variable size packet. If that bit is 1, the related cell is EoP. Otherwise, it is not EoP.

- Packet's size: if EoP bit in the fixed-size cell header is set, in that fixed-size cell header packet's size field represents the length of the variable size packet to which that cell belongs. It is defined in terms of fixed-size packet length. If EoP bit is not set in the fixed-sized cell header, the packet size field is irrelevant. It can be anything.

- Cell's input address: it contains same information with the packet's input address field in variable size packet header. While producing new fixed-size cell header, that field is copied from the variable size packet header.

- Cell's entering time: While producing new fixed-size cell header, that field is copied from the variable size packet header. Thus, it contains the entering time of the variable size packet header to which fixed-size cell belongs.

4.1.3 Brief Explanation of One Time-slot

At the beginning of the time slot, the new variable size packets are welcomed by the input line blocks. Depending on the variable size packet header, the packet size of the packets are determined. According to packet size, new fixed-size cell headers are produced inside the input line block. Then, produced fixed-size cell headers are written to VOQs depending on their class types and destination addresses. Input line blocks send request to fabric scheduler. The fabric scheduler examines the requests and decides the input/output matching decision for the current time slot. Depending on the decision of the fabric scheduler, the switch fabric is set and VOQs in input line blocks are read. After setting the switch fabric, fixed-size cell headers are transferred from input line blocks to corresponding output line blocks. In the output line blocks, fixed size cell headers are investigated whether they are end of packet (EoP) or not.

If a fixed-size cell header does not belong to an EoP cell, it is not required to store. If a fixed-size cell header belongs to an EoP cell, it is stored in a FIFO queue. The EoP cell header which is stored in FIFO can also be treated as variable size packet header since fixed-size cell header which belongs to EoP cell is same with the variable size packet header except 27^{th} bit(EoP) which is unimportant in the variable size packet header(Table 4.1). Then, when the output line is physically available, the variable size packet header in the FIFO queue is read and transferred to the output line. At that time, payload part of the packet is assumed to be read from queue in which it is stored. The amount of the payload data is decided by examining the header since header includes the information of packet's size. That process is repeated for each time slot.

4.2 The Time Slot Trigger Block

Time slot trigger block informs the other blocks of the beginning of the time slot period. The time slot trigger block is the master of the all processes. There are two input signals of the time slot trigger block which are clock and reset. The time slot trigger block produces a signal which is high for one clock duration in each time slot as it can be seen in Fig. 4.3.

Figure 4.3: Time slot trigger block simulation screen

There is a counter inside that block and counter's value is compared with a pre-defined value. The pre-defined value is calculated depending on the clock frequency and time-slot period. When the value of the counter is reached to pre-defined value, output signal is produced as a one-clock pulse. The output signal is connected to *request_generate_command* pin of input line blocks and *control_queue_cmd* pin of the output line blocks. It triggers all input line blocks and output line blocks at the same time. To this end, time slot trigger block synchronizes all input line blocks and output line blocks for each time-slot.

4.3 The Input Line Block

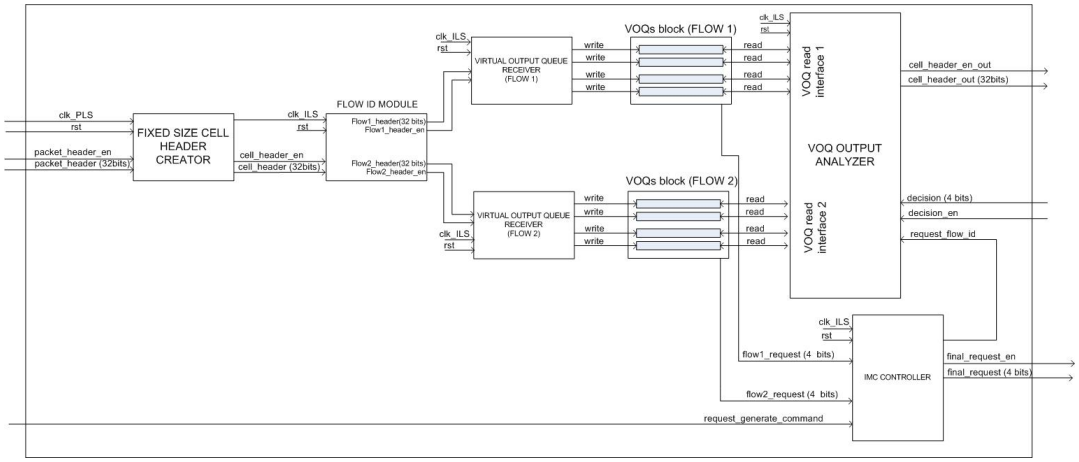

Figure 4.4: General view of an input line block

In the input line block, the variable size packets are welcomed by the fixed-size cell header creator block as shown in Fig. 4.4. The fixed-size cell header creator block checks the packet's size field in the variable size packet header. Depending on the packet's size value, new fixed-size cell headers are produced. For example, if the packet's size equals 2 in the variable size packet header, two fixed-size cell headers will be created. That is, the variable size packet headers are segmented into fixed-size cell headers by that block.

Then, flow id block classifies the incoming fixed-size cell headers depending on their class types. After that block, there are 2 virtual output queue receiver modules as shown in Fig. 4.4. The virtual output queue receiver modules are responsible for writing the fixed-size cells to related VOQs depending on their destination addresses. There are 2 VOQ blocks in each input line block as it can be seen in Fig. 4.4. The number of the virtual output queue receiver modules and VOQs blocks are decided by the number of class types. VOQs are grouped according to the priority level.

Each VOQs blocks has its own requests for the output ports. The IMC controller block decides the final request of the input line block depending on the 2-LLP IMC VOQ algorithm which is explained in chapter 3. The different IMC VOQ algorithms can be implemented in order to provide different QoS requirements. After IMC controller block transmits final request to the fabric scheduler, the fabric scheduler decides the

input-output matching decision. The matching decision is immediately sent to input line blocks. The fabric scheduler is connected to input line blocks as shown in Fig. 4.1. The matching decision informs the input line block about output line block with which it is matched. Moreover, there are multiple queues for each output port. The information about whether flow2 VOQs or flow1 VOQs is going to be read is given to VOQ output analyzer block from IMC controller block. Depending on these two instructions, VOQ output analyzer reads related queue. After it's completing the reading, the fixed size cell header which is read from the queue is transmitted to switch fabric model.

4.3.1 The Fixed-size Cell Header Creator Block

The fixed-size cell header creator block inputs the variable size packet headers. According variable size packet headers, it produces fixed-size cell headers. The fixed-size cell header creator block has 4 input signals 2 of which are clock and reset as shown in Fig. 4.5. *packet_header_en* signal indicates whether *packet_header* signal contains valid data or not, at that time. It is a 1-bit signal. *packet_header* is a 32-bits signal which represents the content of the incoming packet header. There are 2 output signals. At the output, 1-bit *cell_header_en* signal shows validity of 32-bits *cell_header* signal which represents the content of the new produced fixed-size cell headers after segmentation.

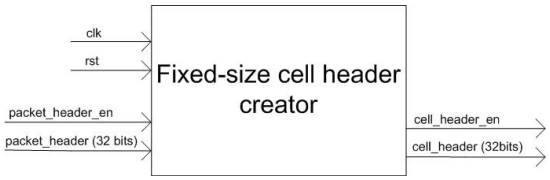

Figure 4.5: Fixed size cell header creator block

When the variable size packet header of the packet is received by fixed-size cell header creator block , the process starts. The whole packet's finishing is not required unlike the store and forward mode as it is done in[28].

In the fixed-size cell header, 27^{th} bit indicates the end of packet. In the last cell of the variable size packet after segmentation, 27^{th} bit is set to 1 in order to indicate

segmentation process is complete and that is the last fixed-size cell of that variable size packet.

According to packet's size field which is defined in variable size packet header, the number of new fixed-size cell headers are determined. When *packet_header_en* signal is active, packet's size field in *packet_header* signal is examined and depending on packet's size field new fixed-size cell headers are produced after 3-clocks delay. That delay is constant for all variable-size packet headers. It does not matter whether the length of the variable size packet equals to one fixed-size cell or to eight fixed-size cells. It is required to analyze packet's size field.

In [28], it is explained that in order to overcome the segmentation overheads, new produced headers must be processed faster than the packet line speed (PLS : packet line speed, ILS: internal line speed).On the other hand, in our implementation, PLS equals to ILS since payload part of the packets are stored in the different RAMs as mentioned in 4.1.

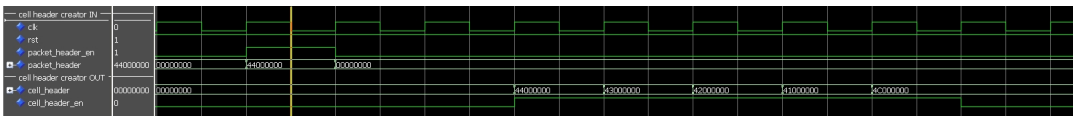

Figure 4.6: Fixed-size cell header creator block simulation screen

In the Fig. 4.6, when *packet_header_en* is active, *packet_header* is 0x44000000. According to variable size packet header format, that header belongs to high priority class since 30^{th} bit is 1. The destination address field in *packet_header* is "00" which states output port 1. The packet's size is 5 which is defined by 26,25 and 24^{th} bits as "100". Thus, at the output of the fixed-size cell header creator block, *cell_header_en* signal is produced as logic *high* for 5 clocks. That means 5 new fixed-size cell headers are produced. The contents of the new produced fixed-size cell headers are represented by *cell_header* signals as 0x44000000, 0x43000000, 0x42000000, 0x41000000 and 0x4C000000. All of them are identically same with the incoming variable size packet header except the EoP(27^{th}) and packet's size(26^{th}-24^{th}) fields. The packet's size information in the first 4 of the fixed-size cell headers will not be used since the cells which are not EoP are discarded at the output line block. Thus, in those cell headers packet's size information is irrelevant as it is explained in 4.2. In

the last cell header which is 0x4C000000, packet's size and EoP bit information will be used at the output line blocks. For the last cell header, EoP bit is 1 which states that the related cell is end of packet. Moreover, in the last cell header packet's size bits are "100" which states a packet size of 5 same with the packet's size field in the original variable size packet header.

4.3.2 The Flow Identification Block

The flow identification block separates the fixed-size cell headers according to their class types. The flow identification block has 4 input signals 2 of which are clock and reset,in Fig. 4.7. 1-bit *cell_header_en* signal and 32-bits *cell_header* signal come from the fixed-size cell header creator block. At the output of flow identification block, there are 4 signals, 2 of them go to one VOQs block storing the cell headers of the high priority class (flow 2), the other 2 of them go to the other VOQs block storing the cell headers of the low priority class (flow 1). $flow2\_header$ and $flow1\_header$ 32-bits signals represent the content of the cell headers after flow identification block. $flow2\_header\_en$ and $flow1\_header\_en$ signals show the validity of the related cell header signals.

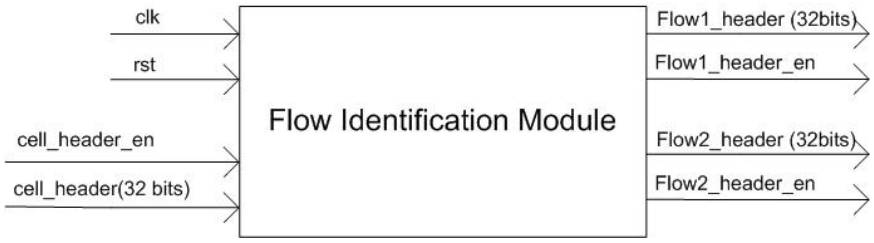

Figure 4.7: Flow identification block

In the fixed-size cell header 30^{th} bit defines the class of the cell. According to classes, cells are identified and transferred to related VOQs blocks.

When *cell_header_en* signal is 1, the cell's class field is examined. Depending on the cell's class field in the fixed-size cell header, $flow2\_header\_en$ or $flow1\_header\_en$ signal is asserted to 1 and incoming *cell_header* signal is transmitted to $flow2\_header$ or $flow1\_header$ signal.

4.3.3 The Virtual Output Queue Receiver

The virtual output queue receiver block separates the fixed-size cell headers according to their destination addresses in order to transfer them to the related queues in the VOQs blocks. As it can be seen in Fig. 4.4, there are 2 virtual output queue receiver blocks in an input port since the implementation supports 2 class types.

The L sign in the signal names represent the class type and L can be 1 or 2 in our implementation. Moreover, the J sign in the signal names represent the output port number and J can be 1,2,3 or 4 in our implementation. These rules are generally followed while naming the signals.

The virtual output queue receiver block has 1-bit $flowL\_header\_en$ signal and 32-bits $flowL\_header$ signals which come from flow identification block. Additional to these signals, clock and reset signals are connected to this block as shown in Fig. 4.8. In our implementation, a VOQs block consists of 4 queues since there are 4 output ports. Thus, there are 4 pairs of 32-bit $flowL\_queueJ\_header$ signal and 1-bit $flowL\_queueJ\_header\_en$ signal at the output of the VOQ receiver block. Each pair is connected to the related queue in VOQs block. For example, $flowL\_queue1\_header$ and $flowL\_queue1\_header\_en$ are connected to the queue 1 in the VOQs block storing the cells of $flowL$ as shown in Fig. 4.4.

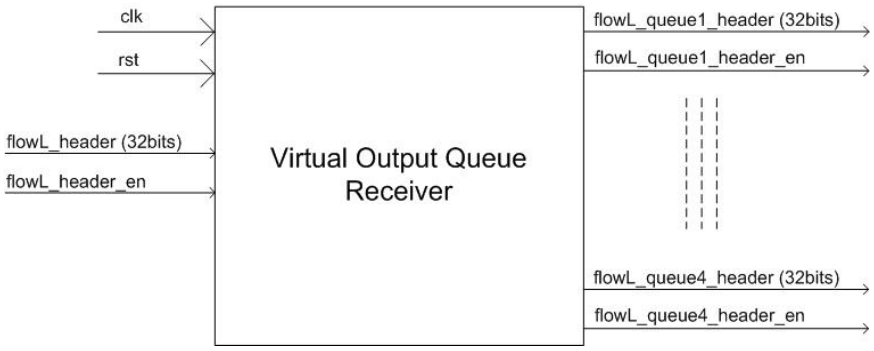

Figure 4.8: VOQ receiver block

In the fixed-size cell header 29^{th} and 28^{th} bits define the destination of the cell. When $flowL\_header\_en$ signal is active, depending on the cell's destination address field, related output signal pair is activated. In this way, cell headers are transferred to the related queue in the VOQs block by the virtual output queue receiver block.

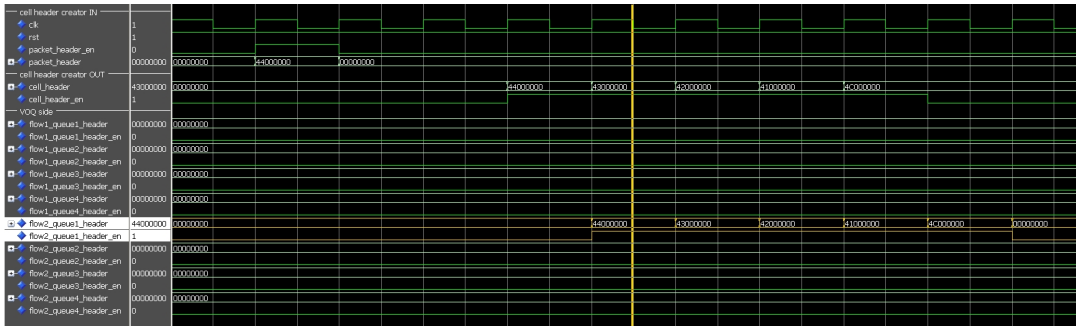

Figure 4.9: VOQ receiver block simulation screen

After segmentation, fixed size cell headers must be submitted to related VOQs. In the Fig. 4.9, new cell headers which are produced by fixed-size cell header creator block such as 0x44000000, 0x43000000, 0x42000000, 0x41000000 and 0x4C000000 can be seen. Here, 30^{th} bits which are all 1 define that these cells belong to the high priority class(flow 2). 29^{th} and 28^{th} bits define the destination port which is output port 1. Thus, all of the produced cell headers are written to queue 1 which is located in the flow 2 VOQs block. In order to write that queue, $flow2\_queue1\_header\_en$ signal is asserted and the cell headers are transmitted through $flow2\_queue1\_header$ signal as shown in Fig. 4.9.

4.3.4 The VOQs Block

The VOQs block stores the fixed-size cell headers in different queues depending on their destination address. In an input line block there are 2 identical VOQs blocks as shown in Fig.4.4. Furthermore, in an input line block, the candidate requests belonging to a priority level is created by the VOQs block.

There are 4 queues in an VOQs block, each one of the queues stores the cell headers which are destined to different output ports. One queue inputs 1 pair of 1-bit write enable signal and 32-bits data signal for write operation. 32-bits $flowL\_queueJ\_header$ signals are connected to write data signals of the queues and 1-bit *flowL_queueJ_header_en* signals trigger the write enable signal of the related queues as seen in Fig. 4.10. On the other hand, $flowL\_queueJ\_rd\_en$ signals are connected to read enable pin of the queues. $flowL\_queueJ\_rd\_data$ signals indicate the content of the data

that is read from the queues. *request_flowL* is a 4-bits signal which represents the candidate request from the VOQs block. The width of that signal equals the number of the queues in VOQs block since each bit of *request_flowL* signal indicates whether there is a request from related queue or not. For example, if there are requests form queue 1 and queue 3, 0^{th} and 2^{nd} bits of the *request_flowL* signal is set to 1, "0101". That is, there are candidate requests for output port 1 and port 3 from that priority level.

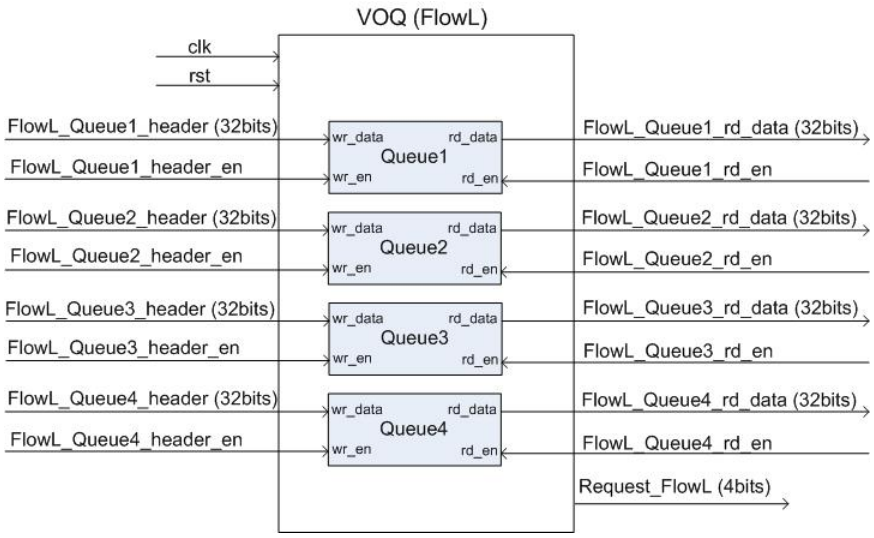

Figure 4.10: VOQs block

The queues are implemented as 1Kx32bits FIFO on FPGA by using LogicCore IP FIFO Generator[40]. The LogicCore IP FIFO Generator can create FIFO blocks by using block RAMs or distributed RAMs inside the FPGA. FIFO blocks have an output signal as empty flag which is asserted high when the FIFO contains at least one data. Thus empty flag can be used to understand whether a queue has a request or not. Thus, 4-bits *request_flowL* signal is concatenation of the empty flags of the FIFOs. On the other hand, the empty flag's being updated by FIFO lasts 2-clocks after the data is written to related FIFO if FIFO is previously empty. Thus, related *request_flowL* signal is updated 2-clocks after the data is written to FIFO.

In Fig. 4.11, *request_flow*2 signal changes its state from "0000" to "0001" which shows that there is a packet in the queue 1 of VOQs block storing the cells of flow2. There is no cell in the VOQs block of flow1 so *request_flow*1 signal remains "0000". The change in *request_flow*2 signal happens 2-clock after the first data is written to

related queue as seen in Fig. 4.11.

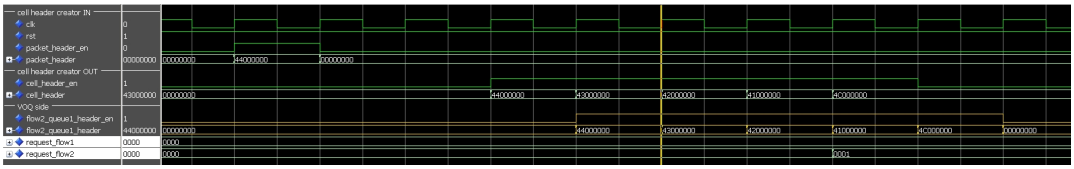

Figure 4.11: VOQs block simulation screen

4.3.5 The IMC Controller Block

There are two candidate requests for each output port in an input line block. The IMC controller block is the intelligent block which decides the request signal which is sent to fabric scheduler from an input line block among those candidate requests. In that implementation, IMC controller block determines the final request depending on the 2-level limited prioritized IMC VOQ algorithm which is explained in chapter 3.

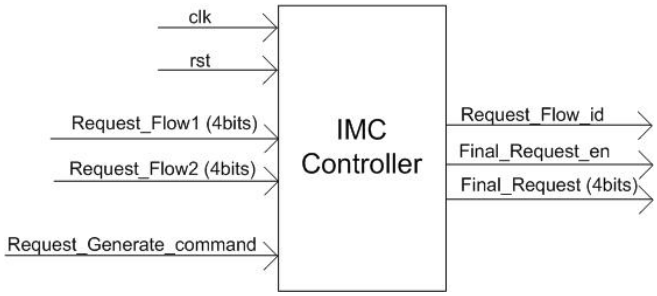

Figure 4.12: IMC controller block

In Fig. 4.12, 4-bits *request_flow*1 and *request_flow*2 signals which come from the VOQ blocks are input signals of IMC controller block. *request_flowL* signals transfer the candidate requests of the VOQs blocks to the IMC controller block. *request_generate_command* which is a one-clock pulse signal is the trigger pin of IMC controller block which starts the process of that block. *request_generate_command* signal comes from the time-slot trigger block and starts the process once for each time-slot period. At the output side, *final_request_en* is the validity signal for 4-bits *final_request* signal which represents the final decided request signal of that input line block. *request_flow_id* signal indicates final request comes from which VOQs blocks, flow 1 or flow 2. In the algorithm 2, that signal is equivalent to the

variable *LastChosenPriorityLevel*.

final_request_en signal is produced as one-clock pulse signal one-clock after *request_generate_command* is active.

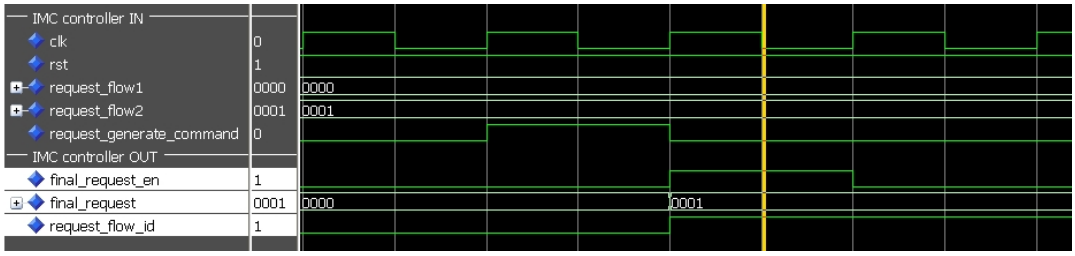

Figure 4.13: IMC controller block simulation screen

In the Fig. 4.13,when *request_generate_command* is active, the request signal from flow1 is "0000" which means that there is not any request for any output ports. On the other hand, the request signal from flow2 is "0001" which means that there is a request for output port 1 from that input port. 2-level limited prioritized algorithm chooses priority level 2 in the Fig.4.13. Then VOQs block of flow 2 is given right to send requests. Thus, final request is asserted as "0001" from VOQs block of flow2. Moreover, when *final_request_en* signal is active, *final_request* signal is "0001" and *request_flow_id* signal is '1'. The *request_flow_id* signal's being 1 shows that final request comes from VOQs block storing the high priority cells(flow 2). That is, 0^{th} bit of the *final_request* signal is set that represents a request to output port 1 from that input port.

4.3.6 The VOQ Output Analyzer Block

The VOQ output analyzer block reads the one of the queues in 2 VOQs blocks depending on the fabric scheduler decision for each time slot. Then, transfers the read data to the switch fabric.

The VOQ output analyzer block contains two read interfaces for two VOQs blocks as shown in Fig. 4.14. 1-bit *decision_en* signal is the validity signal for 4-bits *decision* signal which represents the input-output matching decision of the fabric scheduler for the related input port. Those signals come from fabric scheduler block. Each bit of

the *decision* signal represents the decision of an output port. The number of bit which is 1 in *decision* signal, represents the number of output port which is matched with that input. For example, if *decision* signal is "0100", the output port 3 is matched with the input which is informed with that *decision* signal. *request_flow_id* signal comes from the IMC controller block. *flowL_queueJ_rd_en* signals are connected to the read enable pin of the queues in VOQs blocks. *flowL_queueJ_rd_data* signals represent the data which is read from the queues. *cell_header_en_out* is validity signal for 32-bits *cell_header_out* signal which represents the cell headers which is read from VOQs. *cell_header_en_out* and *cell_header_out* signals are connected to switch fabric module.

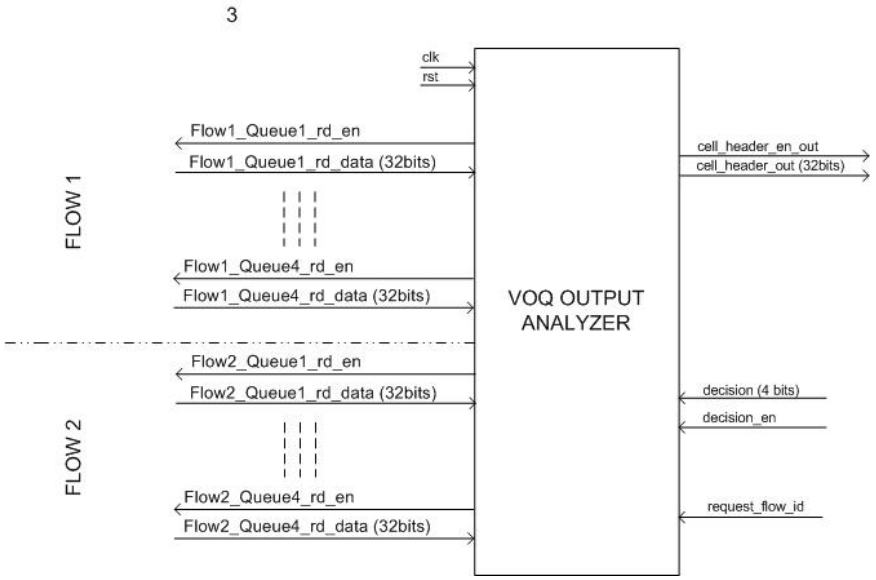

Figure 4.14: VOQ Output Analyzer block

VOQ output analyzer checks the *request_flow_id* signal coming from request analyzer block. *request_flow_id* indicates that queue which is going to be read belongs to flow 1 VOQs block or flow 2 VOQs block. Additional to that signal, 4-bits *decision* signal coming from fabric scheduler is examined by the VOQ output analyzer block. *decision* signal points to the right queue among the queues in the selected VOQs block, flow 1 or flow 2.

After which queue is going to be read is decided, cell header is read and transferred to the switch fabric. The *cell_header_out* signal and *cell_header_en_out* signal are valid 2 clock cycles after *decision_en* is active.

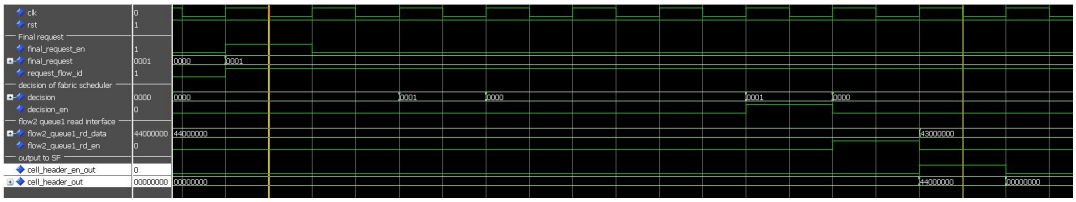

Figure 4.15: VOQ Output Analyzer simulation screen

In the Fig. 4.15 , after *final_request* signal is generated by the input line block, the fabric scheduler decides the matching decision 6 clocks later. The 6-clocks delay of the fabric scheduler block is explained in section 4.4. When *decision_en* signal is 1, the *decision* signal is "0001" which means that output port 1 accepts the request of the input line block receiving that *decision* signal. In the Fig. 4.15, all represented signals belong to the input line block 1 (input port 1). In other words, the *decision* signal's being "0001" means that the fabric scheduler matches input port 1 and output port 1. *request_flow_id* signal represents the class type, 1 means high priority class VOQs(flow 2). Thus, VOQ analyzer block selects flow 2 VOQs by examining *request_flow_id* signal. Then among the queues in VOQs block of flow 2, queue 1 is chosen to read since output port 1 is matched with that input port. After reading the queue, VOQ output analyzer block transfers the cell header to switch fabric block. *cell_header_en_out* and *cell_header_out* signals represent the cell header going to switch fabric module. The cell header which was written to output queue 1 in high priority class VOQs block (0x44000000) can be seen as the cell header going to switch fabric module, in the Fig. 4.15. The field of timer in the cell header is "00000" which states that the packet enters into the router in the time-slot of 0.

4.4 The Fabric Scheduler

The fabric scheduler block which is represented in Fig.4.16 finds a matching between 4 input and 4 output pairs in each time slot by implementing iSLIP algorithm. 1-iteration's result can be obtained after 2 clock cycles in our implementation. Accept and Grant arbiter priority pointers are updated 2 clock cycles after the first iteration is completed. Moreover, our implementation has ability to make second iteration. In order to start the second iteration, request mask signals must be waited. The request

mask signals for further iterations are updated at the same time with priority pointers. If the fabric scheduler is implemented with 2 iterations, it takes 6 clock cycles; 2 clocks for 1st iteration, 2 clocks for request mask signals' update and 2 clocks for 2nd iteration. The iSLIP algorithm does not update arbiter priority pointers after the second iteration. Moreover, the second iteration process use same resources with the first iteration. Thus, it does not increase resource usage dramatically.

The fabric scheduler consists of request creator, request masking, grant arbiters, grant arbiters to accept arbiters, accept arbiters, arbiter pointer updater and combining iterations sub-blocks.

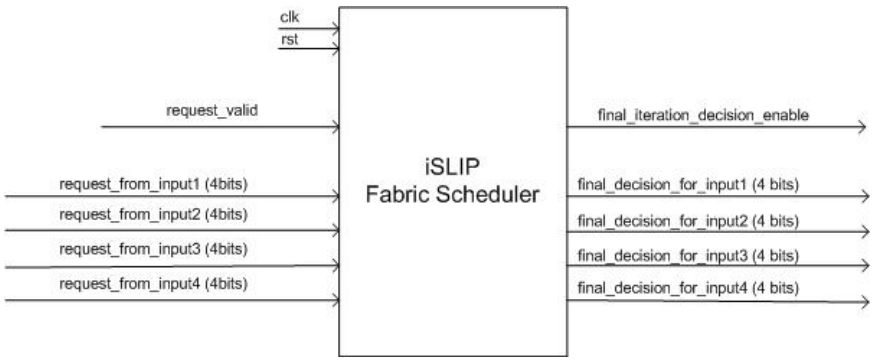

Figure 4.16: iSLIP Fabric Scheduler

The 1-bit *request_valid* signal is asserted high for one clock in each time slot. That signal initiates the fabric scheduler decision process. After completing decision, 1-bit $final\_iteration\_decision\_enable$ signal is driven to high for one clock and the decision of the fabric scheduler is transmitted to input line blocks through 4-bits $final\_decision\_for\_inputI$ signals. Note that, the I sign in the signal names represent the input port number.

4.4.1 The Request Masking Block

The request masking block prevents that matched input and output ports are involved in second iteration. With the help of that block, input-output pairs which are already matched in the first iteration are not included in the second iteration.

The request masking block which can be seen in Fig. 4.17, has inputs as 4-bits

request_from_inputI signals which come from input line blocks outside the fabric scheduler block and 4-bits *request_mask_inputI* internal request mask signals which are produced inside the fabric scheduler block. Moreover, as an output 4-bits *request_for_GrantArbiterJ* signals are distributed to related grant arbiters.

request_from_inputI signals and *request_mask_inputI* signals are logically ANDed by request masking block. *request_mask_inputI* signals are produced for next iterations after the first iteration.

If only one iteration is required, that block can be discarded from the design. Request masking block does not include reset and clock signals thus it can be said that the request masking block is designed as a combinational circuit.

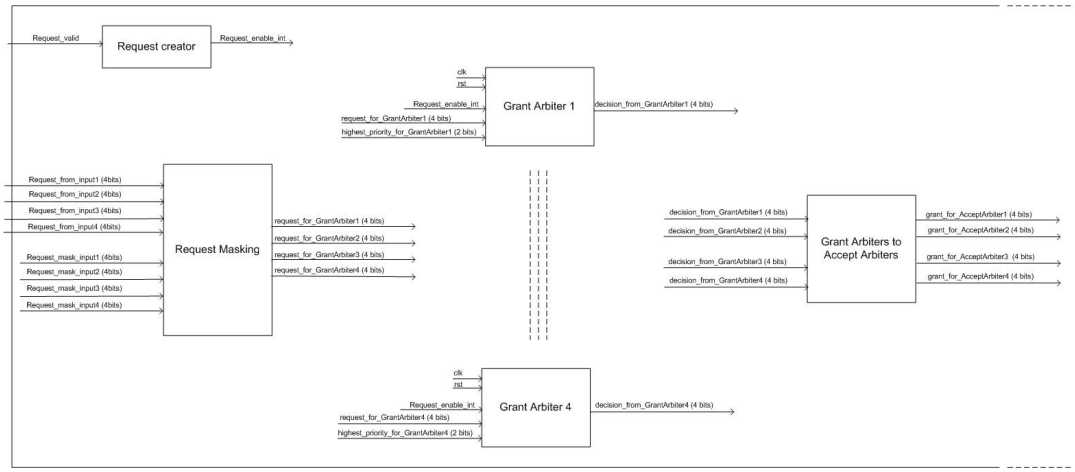

Figure 4.17: iSLIP Fabric Scheduler detailed-1

4.4.2 The Request Creator Block

The request creator block produces the second trigger signal for second iteration. The request creator block has an input as 1-bit *request_valid* signal which comes from input line blocks. Moreover, it has an output 1-bit *request_enable_int* signal which is connected to grant arbiter blocks, in Fig. 4.17.

request_valid signal which comes from input line blocks begins the fabric scheduler process. After the first iteration is completed, in order to start the second iteration, internal trigger signal must be produced. *request_enable_int* signal triggers the grant

arbiters and fabric scheduling process. 2 clocks for first iteration and 2 clocks for mask signals' being ready should be waited for the second iteration. To this end, when external *request_valid* signal is high, *request_enable_int* signal is asserted to high in order to start the first iteration. Then, 4 clocks later *request_enable_int* signal is asserted to high for second time in order to induce the second iteration.

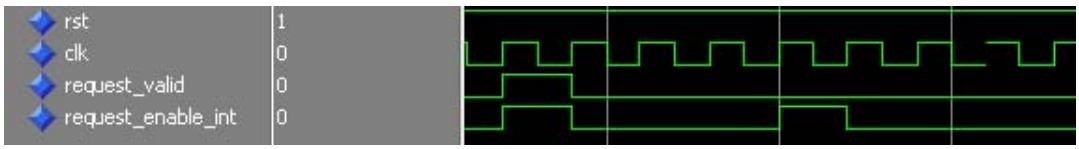

Figure 4.18: The request creator block simulation screen

In the Fig. 4.18, *request_enable_int* signal is asserted high at the same time with the *request_valid* signal for the first time. Then, it is asserted high again after 4-clocks delay. To this end, the grant arbiters and fabric scheduling process are triggered for 2 times (for 2 iterations) in each time slot.

4.4.3 The Grant Arbiter Block

The grant arbiter block chooses the request which is closer to the highest priority pointer. The grant arbiter blocks have inputs as 1-bit *request_enable_int* coming from request creator block, 4-bits $request\_for\_GrantArbiterJ$ coming from the request masking block and 2-bits $highest\_priority\_for\_GrantArbiterJ$ signal which shows the priority pointer for related arbiter, in Fig. 4.17. Depending on the input signals, 4-bits $decision\_from\_GrantArbiterJ$ signal which is connected to accept arbiters will be produced.

The grant arbiter block consists of 3 sub-blocks such as sampling unit, programmable priority encoder and decoder blocks. These sub-blocks are connected to each other as seen in Fig. 4.19.

When *request_enable_int* signal is active, the block starts and the overall process of the grant arbiter block takes 1 clock period for buffering by the sampling unit plus the propagation delay because of the combinational logic of the programmable priority encoder and the decoder blocks. The input signals are buffered by the sampling

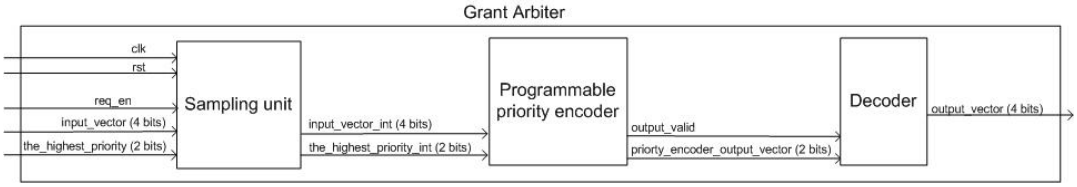

Figure 4.19: The sub-blocks of the grant arbiter block

unit since the rest of the blocks work asynchronous. The buffering input signals results in 1 clock delay. Then, the programmable priority encoder and the decoder blocks which are designed as combinational circuit result in low propagation delay when it is compared with the clock period. *request_enable_int_bufx*1 signal is 1-clock buffered form of the *request_enable_int* signal which is valid signal of the *decision_from_GrantArbiterJ* signal, the decision signal of the grant arbiter block.

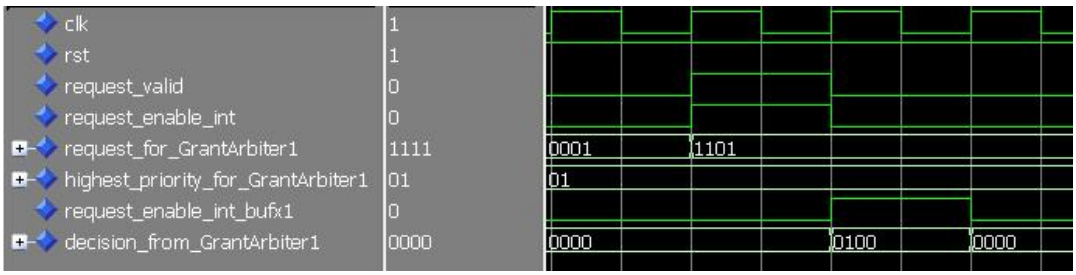

Figure 4.20: The grant arbiter simulation screen

In Fig. 4.20 , *request_for_GrantArbiter1* signal is "1101" which means that there are requests from input 1, input 3 and input 4. *highest_priority_for_GrantArbiter1* signal is "01" which means input 2 has highest priority at that time. To this end, input 3 has the highest priority among the inputs which sent requests. Thus, *decision_from_GrantArbiter1* signal is "0100" which states that input port 3 is granted, at that time. When *request_enable_int_bufx1* signal is asserted to high, *decision_from_GrantArbiter1* signal is ready.

4.4.3.1 The Sampling Unit

When *req_en* signal is active, *input_vector* and 2-bits *the_highest_priority* signals are sampled and transferred to output signals. The other sub-blocks are designed as combinational circuit thus the input signals of the grant arbiter block must be sampled

when *req_en* signal which shows the validity of input signals is 1. In this way, input signals' changing state when validity signal is not active, does not affect the rest of the process.

4.4.3.2 The Programmable Priority Encoder Block

The programmable priority encoder block searches bits of the 4-bits *input_vector_int* signal if it is 1 or 0, starting from the bit which is defined by 2-bits *the_highest_priority_int* signal in ascending order. When it hits the first 1, the number of that bit is displayed as 2-bits *priority_encoder_output_vector* signal in the output together with the *output_valid* signal. If there is not any 1 in the *input_vector_int* signal, the *output_valid* signal will not be asserted to high.

When *input_vector_int* signal does not contain any 1, the value of the *priority_encoder_output_vector* signal is defined as *highZ*. Thus, *output_valid* signal is necessary in order to understand when the *input_vector_int* signal is all 0 since *highZ* signal can trigger the decoder block in unpredictable way.

4.4.3.3 The Decoder Block

The decoder block checks *output_valid* signal. If that signal is 0 which means *priority_encoder_output_vector* signal is not valid, *output_vector* will be "0000". If *output_valid* signal is 1 which means related signal is valid, *priority_encoder_output_vector* signal will be decoded as *output_vector*.

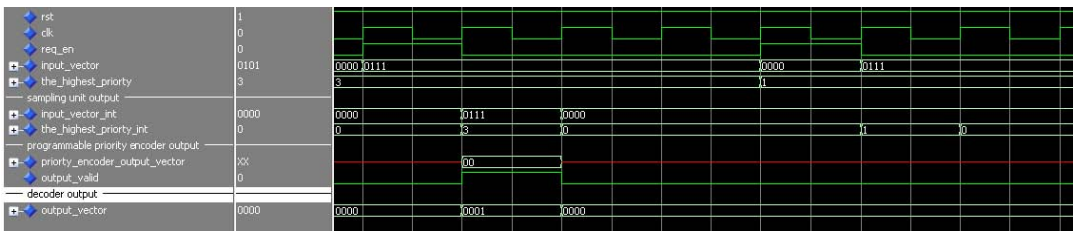

Figure 4.21: The decoder block simulation screen

In Fig. 4.21 , when *req_en* signal is active for the first time, *input_vector* is "0111" and *the_highest_priority* signal points to "3". Depending on these signals, pro-

grammable priority encoder block outputs "00" since 3rd bit is '0' and the closest '1' in ascending order exists in the 0th bit. Moreover, decoder block decodes "00" signal as "0001".

When *req_en* signal is active for the second time, *input_vector* is "0000". Thus encoder block does not make *output_valid* signal is active. If the *output_valid* does not exist in the design, when *input_vector* is "0000", programmable priority encoder block gives output as "XX" (don't care). Therefore, the decoder block will give undefined output depending on its input signal "XX" if *output_valid* signal is not used.

4.4.4 The Grant Arbiters to Accept Arbiters Switch Block

This block transfers the decisions from grant arbiters to related accept arbiters.

The grant arbiters to accept arbiters switch block which has inputs as 4-bits *decision_from_GrantArbiterJ* and outputs as 4-bits *grant_for_AcceptArbiterI* can be seen in Fig. 4.17.

Each bit of the *decision_from_GrantArbiterJ* signal represents an input line block. When grant arbiter *J* grants an input line block, the related bit in the *decision_from_GrantArbiterJ* signal will be high. On the other hand, each bit of *grant_for_AcceptArbiterI* signal represents an output line block. When there is a grant from an output line block for input line block *I*, related bit in the *grant_for_AcceptArbiterI* signal will be high.

While connecting the outputs of the grant arbiters to the inputs of the accept arbiters, it is required that the *decision_from_GrantArbiterJ* signals must be arranged as *grant_for_AcceptArbiterI* signals.

The grant arbiters to accept arbiters switch block is designed as a combinational circuit. Thus, the output signals are produced in a period which is shorter than a clock cycle. Thus, *request_enable_int_bufx1* signal which is synchronous with *decision_from_GrantArbiterJ* signals is still synchronous with *grant_for_AcceptArbiterI* signals which are output signals of that block.

Consider that, the outputs of the grant arbiter blocks are connected to inputs of accept

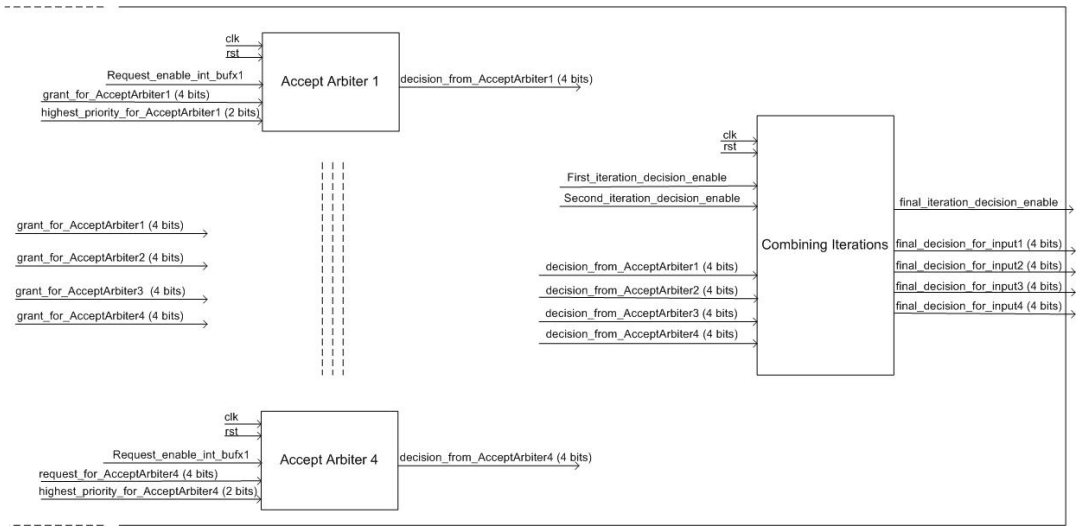

Figure 4.22: iSLIP Fabric Scheduler detailed-2

arbiter blocks through a combinational grant arbiters to accept arbiters switch block. Thus, the buffering structure at the input of the arbiter blocks are also useful for decreasing the combinational path length.

4.4.5 The Accept Arbiter Block

The accept arbiter blocks in Fig.4.22 have the same structure with the grant arbiter blocks in section 4.4.3. The decision takes 1 clock cycle like the grant arbiter blocks. *First_iteration_decision_enable* signal is 1 clock buffered form of *request_enable_int_bufx1* signal which can be used as validity signal with *decision_from_AcceptArbitersI* which are the decision signals of the accept arbiter blocks. *decision_from_AcceptArbitersI* signals also show the final decision of the iSLIP fabric scheduler for related time-slot.

Each bit of the *decision_from_AcceptArbitersI* signal represents an output line block. When accept arbiter *I* accepts an output line block, related bit in the *grant_for_AcceptArbiterI* signal will be high.

4.4.6 The Arbiter Pointer Updater Block

According to the decision of the first iteration, arbiter priority pointers are updated and request mask signals are generated. Request mask signal is used in the second iteration in order to define the matched pairs in the first iteration. To this end, the arbiter pointer updater block is responsible for producing mask signals and updating the arbiter pointers.

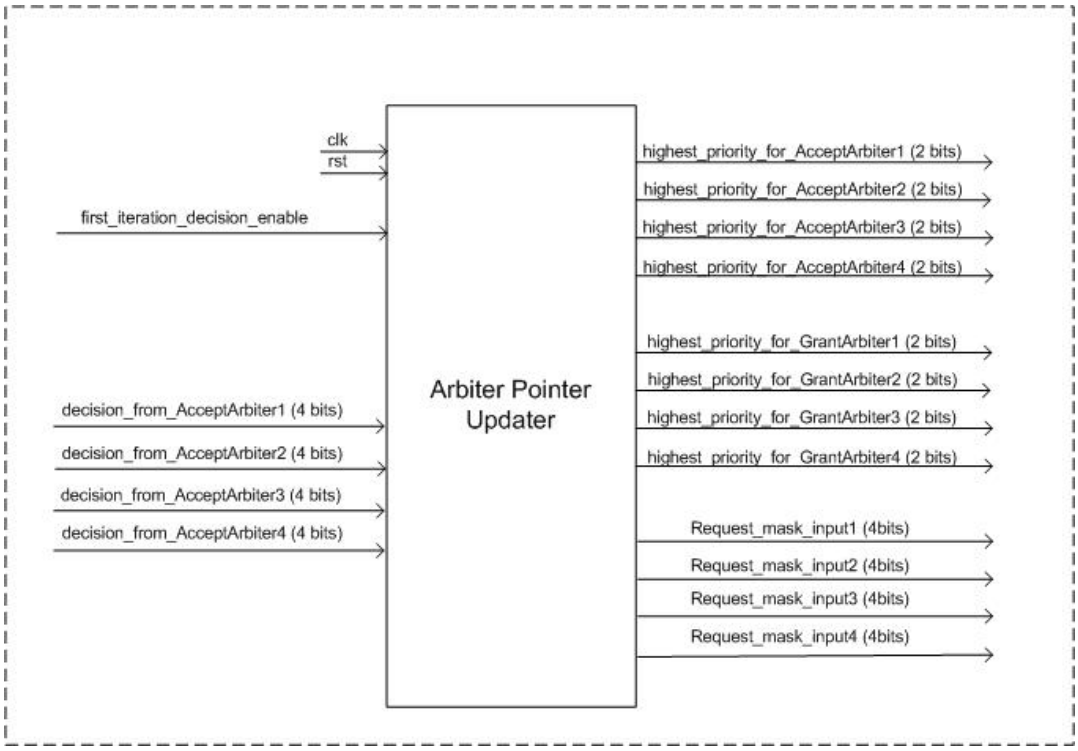

Figure 4.23: The arbiter pointer updater block

The arbiter pointer updater block in Fig.4.23 analyzes 4-bits *decision_from_AcceptArbiterI* when 1-bit *first_iteration_decision_enable* signal is high. These signals represent the matching decision of the iSLIP fabric scheduler after 1st iteration. Depending on the matching decision of the fabric scheduler, four 2-bits *highest_priority_for_AcceptArbiterI*, four 2-bits *highest_priority_for_GrantArbiterJ* and four 4-bits *requset_mask_inputI* signals are produced.

Each *requset_mask_inputI* signal represents an input line block. Each bit of the *requset_mask_inputI* signal represents an output line block. When an output line block is free for the second iteration, the related bit in *requset_mask_inputI* signal is

asserted to high.

first_iteration_decision_enable signal triggers the arbiter pointer updater block. That block gives output 2 clocks after it is triggered. When outputs of that block is ready, the grant arbiter blocks are triggered for the second iteration by request creator block. After the second iteration, arbiter priority pointers are not updated. Thus, only *first_iteration_decision_enable* triggers that block.

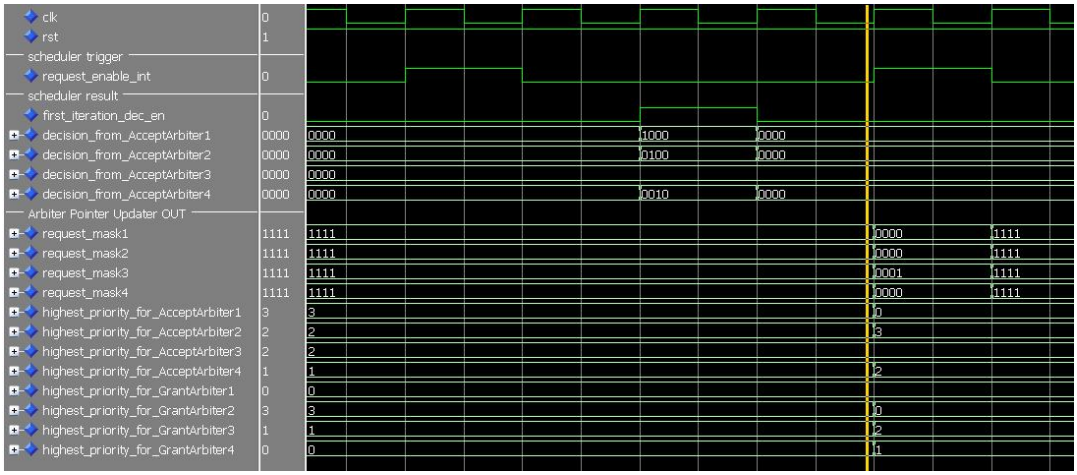

Figure 4.24: The arbiter pointer updater simulation screen

In the Fig.4.24, when *first_iteration_decision_enable* signal is active, the decisions of the accept arbiters are "0010", "0100", "0000" and "1000" which shows that input port 1 accepts output port 2, input port 2 accepts output port 3, input port 3 does not accept any of output ports and input port 4 accepts output port 4. After the first iteration is decided, with a 2-clocks delay, priority signals and request mask signals are updated as seen in Fig.4.24. After the first iteration, among input ports only input port 3 is free. Thus, request mask for input 1,2 and 4 are all 0. Moreover, among output ports only output port 1 is free. Thus, in the *request_mask*3 signal only 0^{th} bit representing the output port 1 is set to 1. With those *request_maskI* signals, in the second iteration, only request from input port 3 to output port 1 is allowed if there is a request.

4.4.7 The Combining Iterations Block

The combining iterations block is required to combine the matching decisions of the first iteration and other iterations if there are any. If only one iteration is supported, that block will be discarded from the design.

After the first iteration is completed, the request creator block produces another trigger signal for the second iteration. The matched pairs are not included in the second iteration. Thus, results of the each iteration must be stored independently and then they must be combined at the end.

The combining iterations block has inputs as *first_iteration_decision_enable* signal, *second_iteration_decision_enable* signal and 4-bits *decision_from_AcceptArbiterI* as seen in Fig.4.22. There is an output as 1-bit *final_iteration_decision_enable* signal which shows validity for the other output 4-bits *final_decision_for_inputI* signals. These output signals go outside the fabric scheduler block and they are connected to related input line blocks.

The combining iterations block stores the result of the first iteration when *first_iteration_decision_enable* signal is active. When the second iteration is complete, whole results are combined by that block. *final_iteration_decision_enable* signal defines that final input-output matching decision of the scheduler is ready.

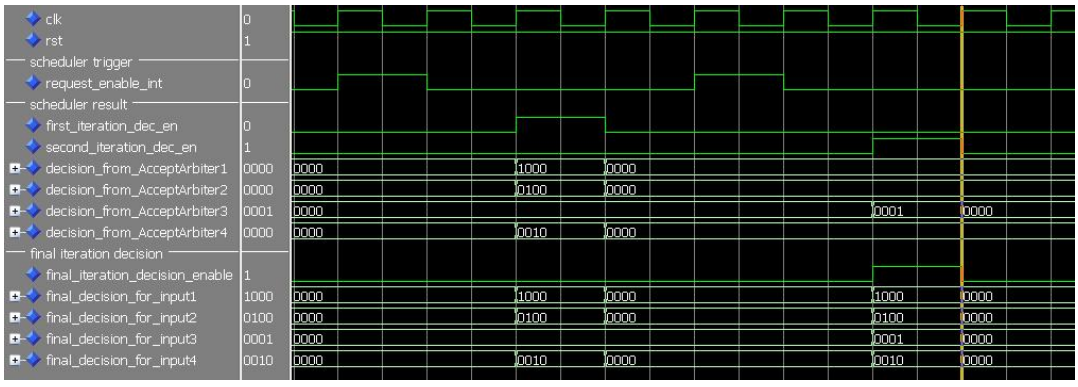

Figure 4.25: The combining iterations block simulation screen

In Fig. 4.25, after the first iteration input 1 and output 4, input 2 and output 3, input 4 and output 2 are matched according to the *decision_from_AcceptArbiterI* signals and the *first_iteration_dec_en* signal. After the second iteration, it is seen that input

3 and output 1 is also matched according to *decision_from_AcceptArbiterI* signals and *second_iteration_dec_en* signal. The combining iterations block combines the results of these 2 iterations as final decision. Then, *final_iteration_decision_enable* signal shows that *final_decision_for_inputI* signals are valid when it is asserted to high(logic '1').

4.5 The Switch Fabric Block

The switch fabric block models the switch fabric. In the FPGA design, according to the matching decision of the fabric scheduler, the connections between matched input and output ports will be realized by switch fabric module in each time slot. Then, cell headers which are read from VOQs in input line blocks are transferred to related output line blocks through the switch fabric block.

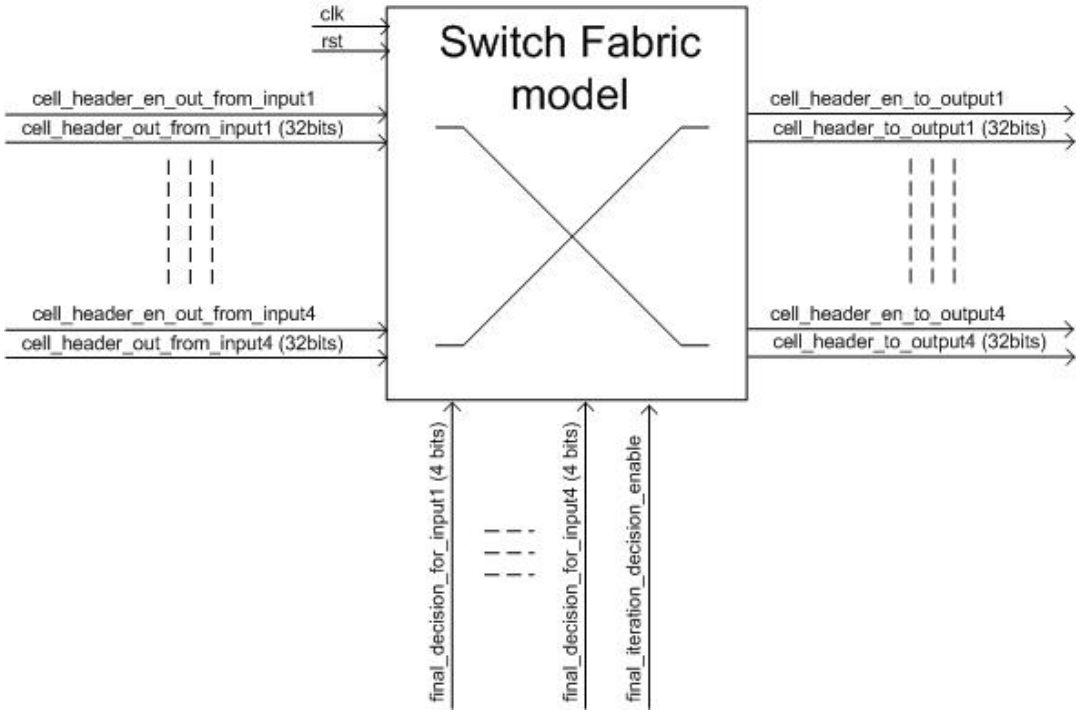

Figure 4.26: The switch fabric block

4x4 crossbar switch fabric is designed depending on the mux-based crossbar switch architecture in [39]. In this structure, all input ports are connected to each output port through a multiplexer. In our design, a pipeline register is inserted at the output of

each multiplexer in order to minimize critical path delay as it is advised in [39].

The interfaces of the switch fabric block are represented in Fig.4.26. 1-bit *final_iteration_decision_enable* signal and 4-bits *final_decision_for_inputI* signals come from the fabric scheduler block. Depending on those signals, the connections inside the switch fabric block is made for each time-slot. 32-bits *cell_header_out_from_inputI* signals and 1-bit *cell_header_en_out_from_inputI* signals come from input line blocks. Moreover, 32-bits *cell_header_to_outputJ* signals and 1-bit *cell_header_en_to_outputJ* signals go to output line blocks.

After the switch fabric configuration is done, transferring from input line blocks to output line blocks takes 1 clock cycle due to the pipeline registers at the end of the switch fabric block.

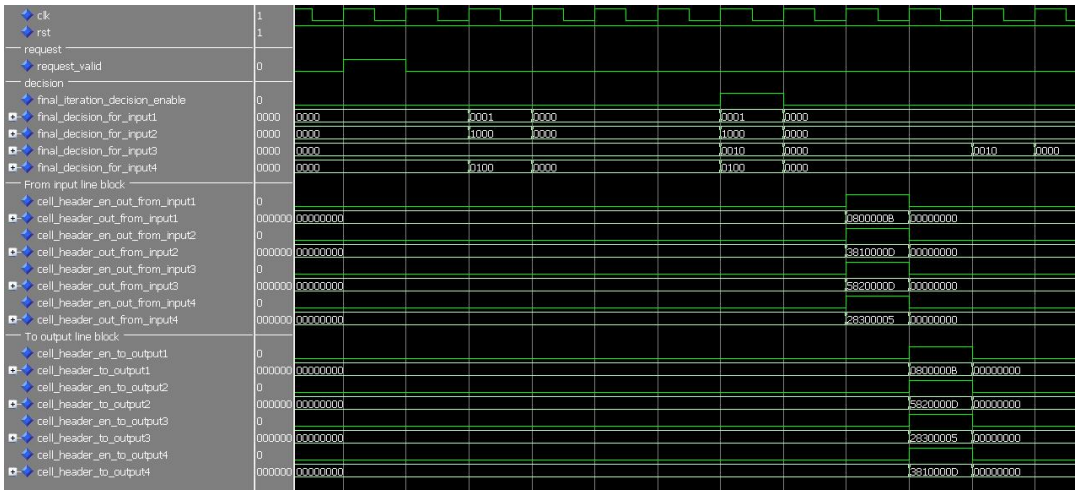

Figure 4.27: The switch fabric block simulation screen

In the Fig.4.27, when *final_iteration_decision_enable* signal is active, according to the *final_decision_for_inputI* signals, input 1 and output 1, input 2 and output 4, input 3 and output 2, input 4 and output 3 are matched. According to the final decision, VOQs in the input line blocks are read. The switch fabric block transfers the cell headers from the input line blocks to the output line blocks with one clock delay as seen from the Fig.4.27.

The cell header which is read form the input line block 1 is 0x0800000B. Depending on the cell header format in Table4.2, that is a flow 1 cell whose destination is output port 1. Thus, related cell header is written to output 1, one clock after that it is read

from input line block.

The cell header which is read form the input 2 is 0x3810000D. Depending on the cell header format, that is a flow1 cell whose destination is output port 4. The related cell header is written to output port 4, one clock after that it is read from input line block.

The cell header which is read form the input 3 is 0x5820000D. Depending on the cell header format, that is a flow2 cell whose destination is output port 2. The related cell header is written to output port 2, one clock after that it is read from input line block.

Moreover, the cell header which is read form the input 4 is 0x28300005. Depending on the cell header format, that is a flow1 cell whose destination is output port 3. The related cell header is written to output port 3, one clock after that it is read from input line block.

4.6 The Output Line Block

The output line block is designed to handle with the variable-size packets. It is responsible for reassembling of fixed-size cell headers into the original variable size packet headers. Thus, if the implementation does not support variable-size packets, there is no need to output line block and it can be discarded from the design.

The output line block welcomes the fixed-size cell headers which are represented by signal pairs of 1-bit *cell_header_en* validity signal and 32-bits *cell_header* signal coming from the switch fabric block. The output line block has *control_queue_command* signal which comes from time-slot trigger block in Fig.4.28. The output signals of that block 32-bits *packet_header_out* signal and 1-bit *packet_header_en_out* signal which represent the variable packet headers which are sent out from the router.

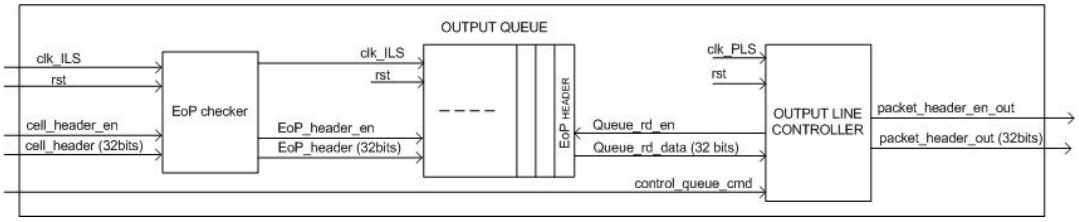

Figure 4.28: The output line block

At the beginning, the incoming cell headers are welcomed by EoP checker block which checks whether cell headers are EoP or not. Then, EoP cell headers are stored in a queue. When output line is physically available, the packet headers are read from the queue and they are placed on the output line. The output line block is mainly designed in order to reassemble variable-size packets which are segmented into fixed-size packets at the input line block.

4.6.1 The EoP Checker Block

The End of Packet checker block decides that cell headers are End of Packet or not. If the cell headers does not belong to EoP cells, they are not transmitted to output of the EoP checker block;in other words, they are eliminated.

EoP checker block has inputs 32-bits *cell_header* signal and *cell_header_en* signal. 1-bit *EoP_header_en* and 32-bits *EoP_header* signals are the outputs of the EoP checker block as seen from Fig.4.28.

The EoP checker block checks the EoP field in the cell headers, when validity signal *cell_header_en* is high. The *EoP_header_en* signal is not asserted to high for cell headers which are not EoP. In that way, the cell headers which are not EoP are not submitted to the output queue block. The process of EoP checker block takes 1 clock cycle.

The EoP cell headers include all necessary information such as packet's size about variable-size packets. Thus, the other cell headers belong to same variable-size packets are not required to be stored in the queue. Hence, only EoP cell headers are written to output queue block.

Figure 4.29: EoP block simulation screen-1

In Fig. 4.29, at the input side, the *cell_header* signal is 0x4C000000 when the *cell_header_en* is high. The EoP field of the cell header states that the cell header is EoP. Thus, it is submitted to the output of the EoP checker block with a 1-clock delay. One clock after *cell_header_en* is high, *EoP_header_en* signal is asserted high and the content of the *cell_header* signal is copied to the content of the *EoP_header* signal.

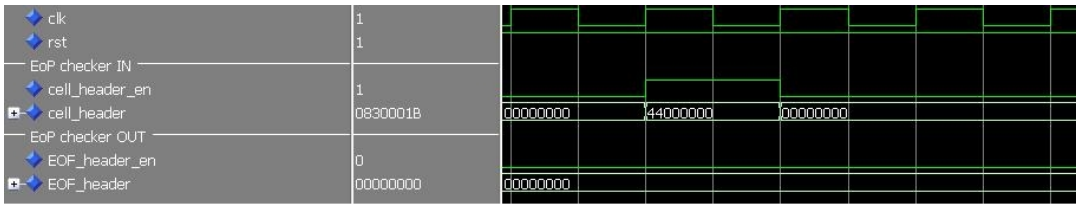

Figure 4.30: EoP simulation screen-2

In Fig. 4.30, at the input side, the cell header is 0x44000000 when the cell header is valid. At the output side, *EoP_header_en* signal is not asserted to high after 1 clock cycle since the cell header is not EoP. In other words, the cell header is discarded since the EoP field in the cell header is not set to 1.

4.6.2 The Output Queue Block

The output queue block stores the cell headers which are EoP cell. The output queue is implemented as 1kx32 bits FIFO on FPGA with same procedure with the queues in VOQs blocks(Section 4.3.4).

The EoP cell headers are written to the output queue block. The header of a packet and EoP cell header which is the last cell of that packet are same except the EoP field in the fixed-size cell header. On the other hand, the EoP field is not used in variable packet header format. Thus, the written cell headers can be used as variable packet headers without any modifications.

4.6.3 The Output Line Controller Block

The output line controller block reads packet headers from the output queue block and reserves the output line for defined packet size length which is stated in the packet

header. When output line is physically available for new packet sending, the output line controller block reads new packet header from the output queue block.

The output line controller block is directly triggered by 1-bit *control_queue_command* signal which comes from time-slot trigger block outside the output line block. The output line controller block has 32-bits *queue_rd_data* signal represents the content of the packet header which is read from output queue block. 1-bit *queue_rd_en* signal is produced by that block and it is connected to the output queue block in Fig.4.28. Moreover, 32-bits *packet_header_out* signal and 1-bit *packet_header_en_out* signal which represent the packet headers sent out to the output line.

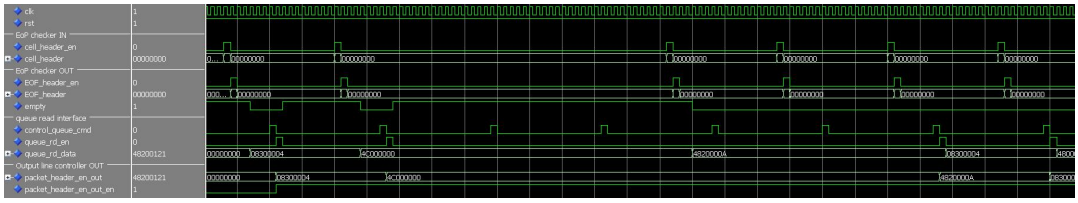

Figure 4.31: output line controller simulation screen

In Fig. 4.31, when the data which is read from the queue is 0x4C000000, the output line controller block holds the output line for 5 time slots and does not read any data from the queue since packet's size field in that header equals to 5. After completing the transfer of the packet whose packet size is 5, output line controller block continues reading next packet header from the queue. While output line controller block holding output line for packet header 0x4C000000, the payload of that variable packet is assumed to be read through payload switch fabric which is shown in Fig4.32.

The payload switch fabric architecture which is shown in Fig.4.32 is assumed to be used to read payload parts of the packets. An output port can decide which RAM is going to be read according to the field of the packet's input address. The field of packet's size gives that how many bytes is going to be read from the queue.

When variable-size packet header is put on the output line, the packet's size and input address information can be acquired from the header. The payload switch fabric is configured depending on the input address. Moreover, packet's size information defines the duration of that configuration. In that way, output port reads and outputs the payload data by reading payload RAMs through payload switch fabric.

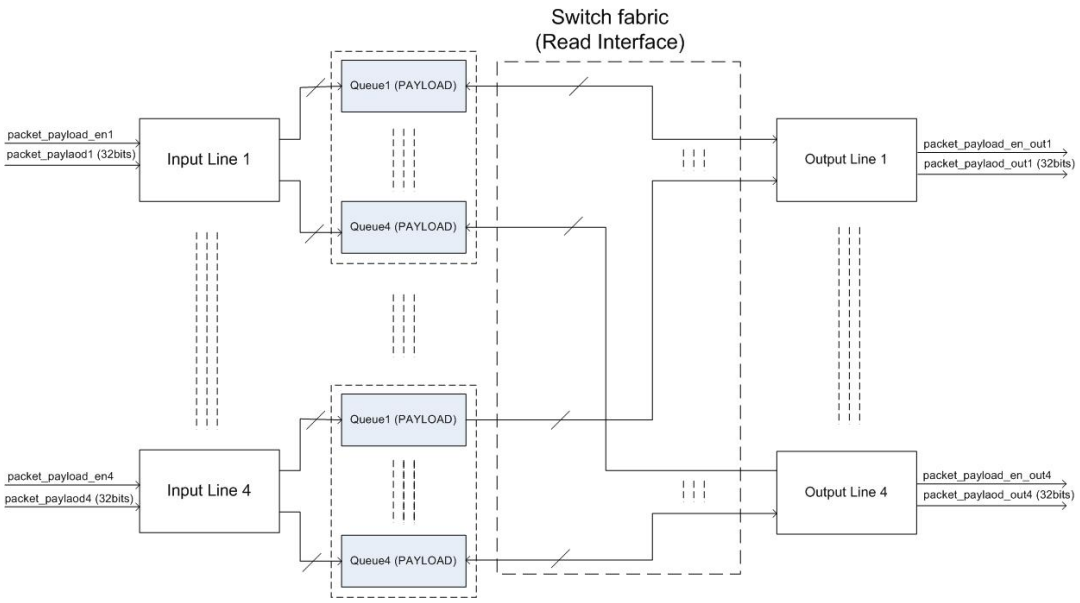

Figure 4.32: General View of payload transfer

The payload RAMs are assumed to be designed in the same structure with VOQs in the FPGA as it can be seen in Fig.4.32. Thus, different output ports are able to read from the same input port at the same time. While working only with fixed size cells, different output ports do not read same input port at the same time. On the other hand, different output ports can read same input port at the same time while supporting variable size packets. With that structure, the packet headers are transferred through the switch fabric inside the FPGA and it is assumed that the packet payloads are assumed to be transferred through second switch fabric outside the FPGA. Instead of increasing hardware speed, the second alternative path is defined for payload transfer.

CHAPTER 5

A GENERAL HARDWARE FRAMEWORK FOR FABRIC SCHEDULER IMPLEMENTATION

The hardware blocks on the data plane are similar and can be reused for designing and implementing different types of switch/router. Thus, previously designed blocks can be used with minor changes in order to implement different routers. In an application-specific integrated circuit(ASIC) design , it is not possible to change the design for different requirements. However, FPGA devices provide programmable hardware design. With minor changes on the VHDL source codes, the port sizes can be increased or decreased. Moreover, the number of priority levels which are supported can be changed. Furthermore, the IMC request selection policy can be altered. Moreover, different fabric scheduling algorithms can be realized. Thus, new routers for different requirements can be easily designed and implemented on the FPGAs by doing minor changes on the VHDL source codes.

In Chapter 4, the FPGA implementation which is designed for 4x4 2-LLP IMC VOQ iSLIP router which supports 2 priority levels is explained in detail. Next, in this chapter, firstly, changing the IMC request selection policy is explained. Secondly, changing the number of the supported priority levels is discussed. Thirdly, changing the number of ports is discussed. Lastly, it is examined to create a switch with different fabric scheduler algorithm such as DRR and PIM. What kind of changes are required on VHDL codes are explained. Moreover, the effects of the changes on FPGA resource usage are discussed.

5.1 Changing the IMC request selection policy

The IMC request selection algorithm is implemented inside the input line block. The IMC controller block which is represented in Fig.4.4 realizes the IMC request selection algorithm. Different request selection algorithms can be realized inside that block. As it is explained in Chapter 3, the request selection approaches can be categorized into 2 groups as separated selection and combined selection. The implemented input line block structure supports all prospective algorithms based on combined selection approach. On the other hand, for algorithms based on separated selection approach, flow id must be remembered for all output ports. Thus, the number of signals which carry flow id from IMC controller block to VOQ output analyzer block must be equal to the number of the output ports. Additionally, the VOQ output analyzer block must be modified in order to evaluate the incoming flow id module for each output port, when separated selection approach is preferred.

5.2 Changing the number of supported priority levels

The multiple priority levels support is implemented inside the input line block (Fig.4.4). If a change occurs in the number of priority levels, the flow id module, IMC controller and VOQ output analyzer blocks are required to be updated. The virtual output queue receiver and VOQ blocks are used without any modification. Only numbers of those blocks change. The number of those blocks must equal to P which is the number of priority levels.

While changing the number of the priority levels, the first change must be done on the flow id module. The flow id module must have capability to separate coming cells into P different interfaces, one interface for one priority level. In order to do that, the designed flow id module must be updated. The same design procedure must be followed for P different interfaces with the flow id block in our FPGA implementation. The flow id module is designed by using 1 to 2 de-multiplexers since the input signal pair is de-multiplexed to 2 output signal pairs depending on the class type, in our FPGA implementation. If there are P priority levels, 1 to P de-multiplexers must be used.

The IMC controller block chooses the final request among P candidate requests come from the P VOQs blocks. The number of candidate request inputs in IMC controller must be changed to P. On the other hand, the resource usage of IMC controller block is tightly depend on IMC VOQ algorithm.

The number of read interfaces must be changed in the VOQ output analyzer block. In that block, there are VOQs read interfaces which are responsible for reading VOQs depending on the destination address. The number of read interfaces must be equal to P.

The rest of the design, the fabric scheduler block, the output line blocks and the switch fabric block can be used without any change.

To sum up, the resource usage of the input line blocks scales with $O(P)$. The other blocks stay constant while changing the number of supported priority levels.

5.3 Changing the number of the ports

In order to design with N input and output ports, the number of input line blocks and output line blocks must be N in the Fig.4.1.

In the input line block (Fig.4.4), the virtual output queue receiver block must be changed in order to write N queues. In order to do that, 1 to N de-multiplexer is used. Moreover, the number of the queues inside the VOQs block must be changed to N. These queues are implemented by using block RAM resources of the FPGA, thus the block RAM usage changes with N in an input line block. The width of the request signal changes to N since there are N candidate queues in each VOQs blocks. Thus, IMC controller block candidate request inputs and final request output must be updated since the width of the request signals changes to N. Moreover, the size of the read interface inside the VOQs output analyzer must be enough to read N queues. To this end, the width of the decision signal must be N since it represents the number of queues in each VOQs block.

In the input line block, the RAMs usage changes with $O(N)$ since N queues are implemented by using block RAMs in the FPGA. The resource usage of a single input line

block changes with $O(N)$. The number of input line blocks changes with N. Thus, total cost of the input line blocks scales with $O(N^2)$.

The request inputs and decision outputs of the fabric scheduler block (Fig.4.16) must be updated in order to support new width of N. The changes in the fabric scheduler block depend on the fabric scheduler algorithm. When iSLIP is implemented, the number of the grant arbiters and accept arbiters must be changed to N. That is, there are N^2 connections between N accept arbiters and N grant arbiters. Furthermore, each arbiter's resource usage scales with $O(N)$ since they give decision from N possible options. Furthermore, iSLIP consists of $2N$ arbiters scaling with $O(N)$. To this end, resource usage of iSLIP fabric scheduler also scales with $O(N^2)$.

The switch fabric consists of N times 1-to-N multiplexers. Thus, it can be said that the switch fabric module resource usage scales with $O(N^2)$.

The output line blocks can be used without any modification. Although the resource usage of a single output line block does not change, total cost of output line blocks scales with $O(N)$ since the number of output line blocks must be updated to N.

To sum up, while changing the number of input and output ports to N, total cost of input line blocks, iSLIP fabric scheduler and switch fabric module scale with $O(N^2)$. On the other hand, total cost of output line blocks which are only necessary for variable size packet support scale with $O(N)$.

5.4 Changing the fabric scheduler

The fabric scheduler blocks have almost same interfaces. Thus, the fabric scheduler blocks can be changed easily. In the Fig. 4.1, it can be seen that the fabric scheduler block has connections with input blocks as request signals and decision signals. The fabric scheduler block also sends the decision signals to switch fabric module.

With those interfaces any other fabric scheduler block can be plugged instead of iSLIP fabric scheduler. For example, DRR[34] and PIM[33] algorithms have same interface with iSLIP. Both of them evaluate the N-bit requests in order to give decision. Thus, these types of fabric scheduler blocks can be placed instead of iSLIP without any

modification on the rest of the design.

On the other hand, fabric scheduler algorithms can decide input-output matching depending on the information from the input lines such as queue length. Our iSLIP implementation only supports 1-bit request information for each output port from an input port. That is, an input line block sends N-bits request signal, each bit of that signal is transmitted to an output port arbiter in the fabric scheduler block. That signal carries the information whether there is a request or not for each output port. In order to send X-bits queue length information for each output port instead of 1-bit request, 1-bit request connections must be updated to X-bits. To this end, the N bits request signals from input line blocks to fabric scheduler must be updated to NxX bits signals. However, the decision signal is kept same as N bits since the decision signal informs input line blocks and it is not related with the way how the fabric scheduler decideds. That is, input-output matching decision can be submitted to input line blocks through same N bits decision signals.

CHAPTER 6

PERFORMANCE EVALUATION

The overall design must be functionally verified at the end of the design procedure. In order to verify that overall design, the VHDL source codes are simulated by using MODELSIM. The necessary signals such as input packets and output packets are written to text files cycle by cycle during the simulation which is carried on MODELSIM. Next, these text files are processed by MATLAB in order to make them more readable since signals are written as binary by MODELSIM. At the end, the results are verified manually. After passing manual verification, the overall design is verified by comparing the results with C++ based simulator which is proposed in [41].

The VHDL codes for complete designs 4x4 basic iSLIP, 4x4 2-LSP IMC VOQ iSLIP and 4x4 2-LCP IMC VOQ iSLIP are produced in the manner which is described in the chapter 5. Then, they are verified by C++ based simulator [41]. In this way, all basic architectures are functionally verified.

The verified design results are used in order to calculate the packet delay and throughput. MATLAB is preferred so that the packet delay and throughput of 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP are evaluated. The extensive simulations can be found in [41].

After functional verification, the resource usage of designs are examined. The overall designs are separately synthesized by Xilinx ISE 12.4 in order to determine the exact FPGA resource usages for each design.

The VHDL codes for complete designs 4x4 basic iSLIP, 4x4 2-LLP IMC VOQ iSLIP and 8x8 2-LCP IMC VOQ iSLIP are produced in a way that is described in the chapter

5. Also, they are synthesized by Xilinx ISE 12.4. Firstly, 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP are compared with an aim to see the effects of changing priority levels as it is discussed in chapter 5. Then, 4x4 2-LLP IMC VOQ iSLIP and 8x8 2-LLP IMC VOQ iSLIP are compared in order to see the effects of changing the number of ports.

Note that, iSLIP algorithm is implemented with two iterations during the whole experiments mentioned above: verification, simulations and resource usage analysis.

6.1 Behavioral Simulation of VHDL codes

Of the three simulation methods (behavioral, structural, and timing), the behavioral simulation runs the fastest but provides the least design information [42]. Behavioral simulation allows you to verify syntax and functionality without the information on timing. During the design development, most of the verification is accomplished through the behavioral simulation. After the required functionality is achieved, the structural and timing simulation methods can be implemented to obtain more detailed verification data. Behavioral simulation is performed by using a pre-synthesis Hardware Description Language (HDL) description of the design [42].

Simulations of FPGA implementation is carried out by using MODELSIM. The simulations are implemented on RTL (behavioral) code, which does not cover timing issues to verify its functionality.

In order to do RTL code simulation, input traffic is applied to the design under the test. There are two ways of applying input traffic. In the first one, a VHDL block generates a poisson traffic instantaneously during the simulation period. In the second one, the packets are written as a text file before the simulation is started on MODELSIM and during the simulation period this text file is read by a VHDL block. Here, the sequence and content of the input packets are known before the simulation.

During the simulation, internal signals such as requests and priority pointers in each time-slot are written to a text file by a VHDL block. At the end of the simulation, that text file can be analyzed in order to verify that the design works correctly.

Moreover, all packets at the input and output of the design are written as another text file by a VHDL block in each time-slot. The text file includes the packets' entering and leaving time information. During the simulation, the size of text files is getting bigger since in each time-slot new packets are added to them.

After completing the simulation on MODELSIM, result text files are processed by using MATLAB. The simulation environment on MODELSIM is not as fast as MATLAB while writing/reading the text files. Thus, the rest of verification work is carried on MATLAB. With the help of MATLAB, the text files are rearranged in order to make them more readable. Then, the functional behavior of the design is verified cycle by cycle manually. Lastly, FPGA design is verified by the C++ iSLIP simulator [41] in more detail by applying the same pre-defined text files which store input traffic to C++ based simulator [41] and FPGA design.

6.1.1 Input Traffic Generation

In order to evaluate the performance of the designs, input traffics must be applied to design under test. A VHDL block is written with the purpose to simulate the input traffics. That block is not synthesized, but rather it is only used for simulation by MODELSIM. The block generates poisson traffic for each input port. The traffic is also uniformly distributed among output ports. The load of traffic can be changed. Moreover, it is also possible to alter the percentage of the high priority class packets and low priority class packets. That block generates the traffic randomly for each experiment run-time. To verify the design with another simulation setup, another VHDL block is written to apply input traffic. That block reads the input traffic from a text file. In this way, the design under test can be verified with a pre-defined data set.

6.1.1.1 Poisson traffic generator VHDL block

The poisson traffic generator VHDL block uses *ieee.math_real* library which consists of a random number generator. On the other hand, that library can not be synthesized in order to be implemented on FPGA, rather it can be used only for simulations.

For each input ports, the traffic loads and the percentage of high priority class and low priority class packets can be defined. Depending on those parameters, the traffic for each input port is generated as follows:

step 1.(initialization) The time-slot counter is used to wait for time-slot period between two packets. Moreover, input port number counter is used to switch to the next input port. Clear the time-slot counter and input port number counter and go to the step 2. This step is a kind of reset state.

step 2.(time-slot period) The time-slot counter is incremented by one for each clock cycle and it is checked whether it is equal to time-slot length value or not. If it equals to the time-slot length value, reset the time-slot counter value to 0 and go to the step 3; otherwise, stay in the step 2 and continue to incrementing time-slot counter.

step 3.(traffic load) A random number between 0 and 1.0 is produced. If that value is smaller than pre-defined traffic load value, in that time-slot the packet header is generated for input port which is defined by input port number counter. Write input port number counter to the input address field in the packet header and go to the step 4. If the produced random number is not smaller, the packet header is not generated for that input port and go to the step 6 in order to switch to the next input port.

step 4.(packet's class) In this step, the class of the produced packet is determined and written to packet header. A random number between 0 and 1.0 is produced. If that value is smaller than the pre-defined value of the percentage of the low priority class packets, the produced packet is generated as low priority class packet; otherwise, it is defined as high priority class packet. The packet's class field in the packet header is written. Then go to the step 5 to determine the destination port.

step 5.(packet's destination address) The produced packets are uniformly distributed among output ports. Thus, a random number is generated between 0 and 1.0 again. If the produced random number is between 0 and 0.25, the destination is output port 1. If it is between 0.25 and 0.5, the destination is output port 2. If it is between 0.5 and 0.75, the destination is output port 3. Lastly, if it is between 0.75 and 1.0, the destination is output port 4. In this way, the input packets are distributed uniformly among the output ports. The defined output port number is written to packet header.

After deciding destination, packet header production for an input port is completed in that step. In order to switch to next input port go to the step 6.

step 6.(next input port) In this step, it is checked if the process is applied for all input ports or not for that time slot. Increment the port number counter. If it equals to the number of input ports, it means that all input ports are processed for that time-slot, clear input port number counter and then go to the step 2 to wait for next time-slot; otherwise, go to the step 3 to apply the same procedure for the rest of the input ports.

Until the simulation is stopped by the user, that VHDL block produces packet headers in each time-slot as defined. The produced packet headers are sent to the design under test simultaneously. The sequence and content of the packet headers are randomly chosen by that block on each run.

6.1.1.2 The packet reader from a text file VHDL block

The packet reader from a text file VHDL block uses another special library *std.textio* which cannot be synthesized. That library is used for reading and writing data on a text file. Different text files are employed for different input ports.

Firstly, it is required to create an input traffic and write it as a text file. Then, the same text file can be re-used for many times. The sequence and contents of the packet headers do not change on each run. In this way, if you have an input traffic and a corresponding output traffic which is verified by another simulation tool, you can compare your output traffic with the verified output traffic by applying the same input traffic.

Each row of the text file represents the packet header in Fig.6.1. Thus, each row consists of 32-bit 0 or 1 in a defined way in Table 4.1. On the other hand, 32 bits of X means that there is no packet for the related time-slot. Each text file represents the traffic for an input port. In each time-slot period, one row of the text file is read.

That block runs as follow:

step 1.(initialization) Clear the time-slot counter and go to the step 2. This step is a kind of reset state.

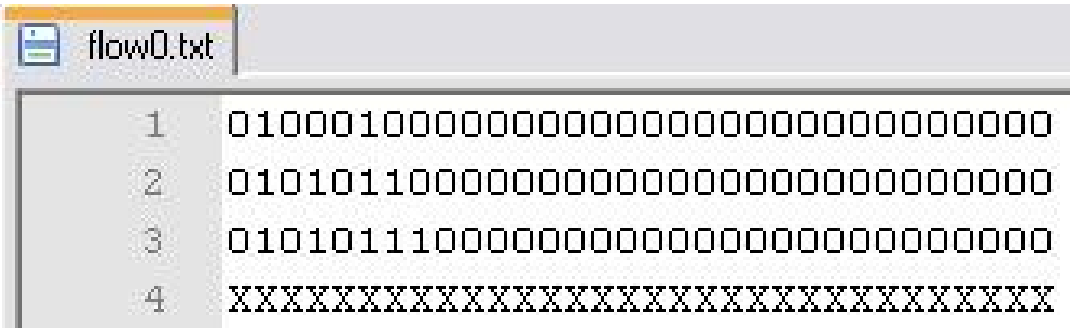

Figure 6.1: Input text file example

step 2.(time-slot period) The time-slot counter is incremented by one for each clock cycle and it is checked whether it is equal to time-slot length value or not. If it equals to the time-slot length value, reset the time-slot counter value to 0 and go to the step 3; otherwise, stay in the step 2 and continue to incrementing counter.

step 3.(End of text file check) In this step, whether the text file is completed or not is controlled. If the last line of text file is reached, go back to the step 2; otherwise, go to the step 4 in order to read the next line.

step 4.(next line) The next line of the text file is read. If the data which is read equals to 32-bits X, the packet header is not produced for that time-slot; otherwise, the read data is submitted to the input port as a packet header. Then go back to the step 2.

That block is written for one input port. Hence, there are different text files for each input ports and different traffic reader blocks are connected to each input port.

6.1.2 Packet recorder

The packet recorder VHDL block uses the same library *std.textio* with the traffic reader from a text file VHDL block. That block records instantaneous time-slot information and also the packets in the related time-slot to a text file. The packet recorder block can be connected to both output ports and input ports. In this way, it can record the input traffic and the output traffic to different text files. The content and sequence of the packets are gathered from those text files.

That block runs as follows:

step 1.(initialization) Clear the port number counter and go to the step 2.

step 2.(time-slot period) Check the time-slot value. If the time-slot changes, the 20-bit time-slot value is written to text file and then go to the step 3; otherwise, stay in the step 2 waiting for the next time slot.

step 3.(port control) The port which is pointed by the port number counter is checked whether there is a packet on it or not. If there is a packet, it is written to the text file; otherwise, go to the step 4.

step 4.(next port) In this step, it is checked if the process is applied for all ports or not. Increment the port number counter, if it equals to the number of ports, it means that all ports are completed for that time-slot, clear port number counter and then go to the step 2; otherwise, go to the step 3 in order to apply the same procedure for the rest of the ports.

The written text file includes all packet headers which exit from all output ports for each time-slot period. In Fig. 6.2, the text file which is written by the recorder block connected to the output ports can be seen. The 20-bit data represents the end of each time-slot, which are circled in red. The other 32-bit data represents the packet headers which exit from the router.

6.1.3 Fabric scheduler logger

The fabric scheduler logger VHDL block uses the same library *std.textio*. The internal communication signals of the fabric scheduler are recorded to a text file. For the iSLIP implementation, the requests from the input signals and arbiter pointers are written to the text file. With the help of that text file, functional verification of the design can be done cycle by cycle.

For the 4x4 iSLIP, there are four 4-bit request signals. Moreover, there are four accept arbiters and four grant arbiters which have 2-bit pointer signals. Thus, in each time-slot 32-bit data is written to the text file.

In the text file in Fig.6.3, each line corresponds to a time-slot value. The first 4 bits represent the request from the input 4, the second 4 bits represent the request

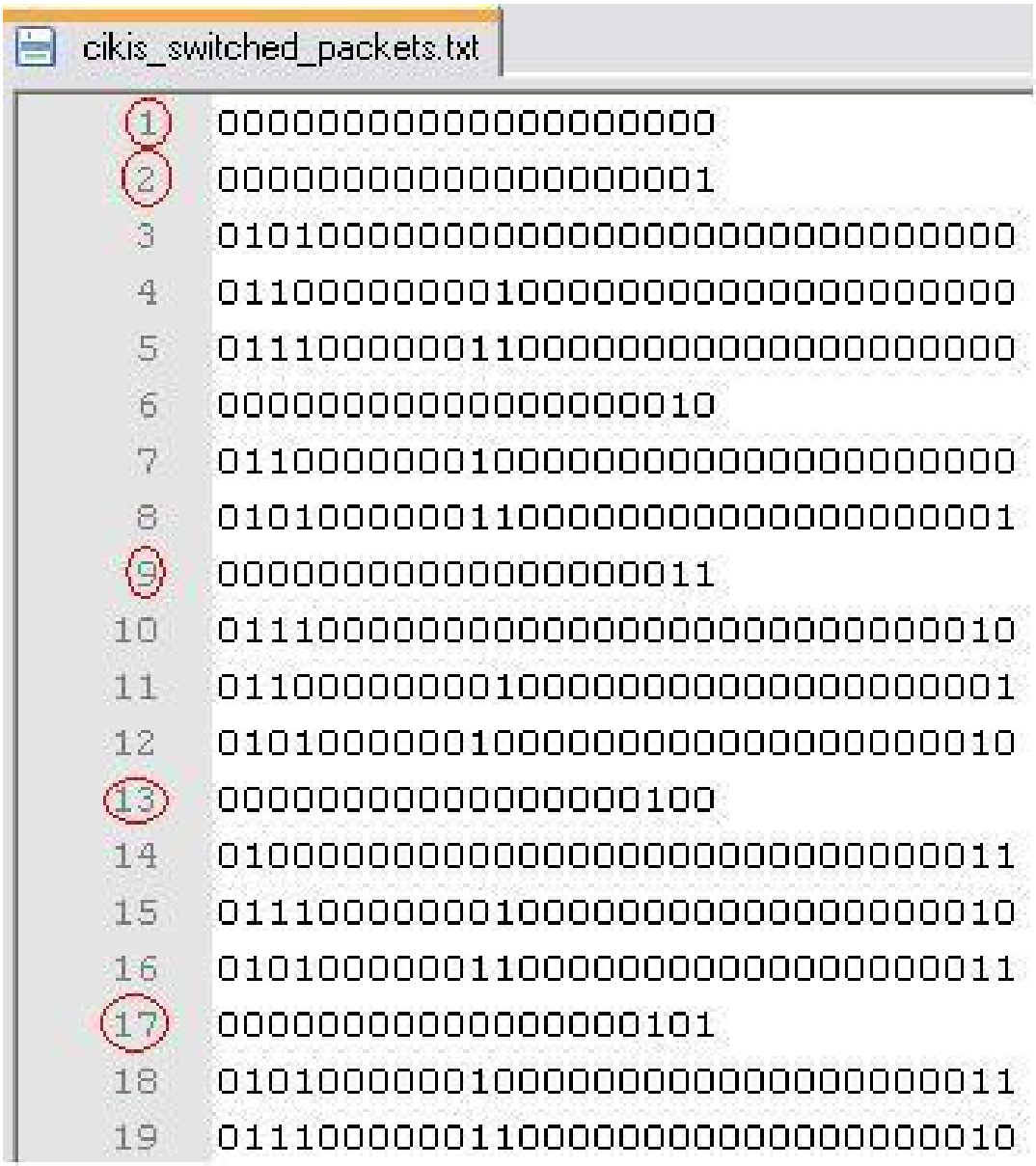

Figure 6.2: Output text file example

from the input 3 and continues in descending order. The initial 16 bits represent the requests, the last 16 bits represent the arbiter pointers. The first 2 bits stand for the priority pointer for the accept arbiter 4, the second one is for the accept arbiter 3 and continues in descending order. The same order is implemented for grant arbiters also in last 8 bits.

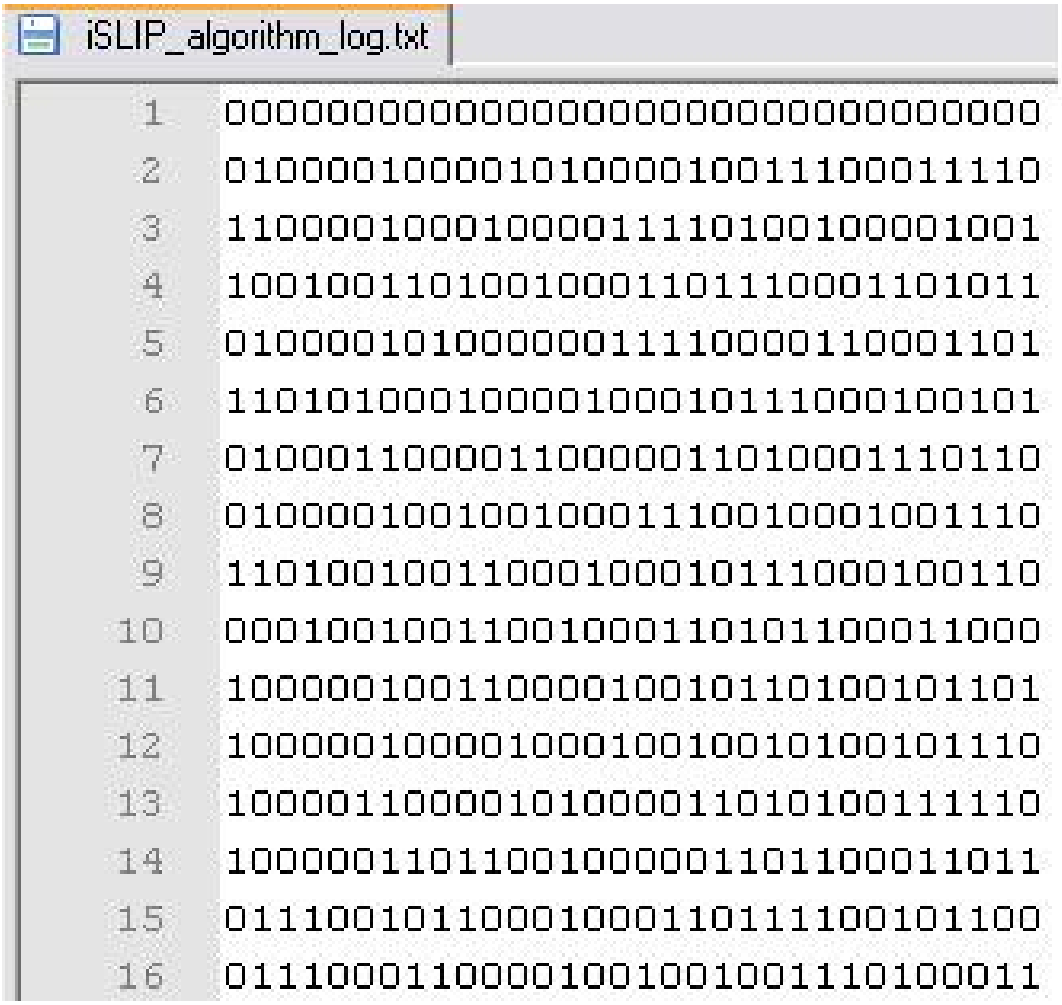

Figure 6.3: iSLIP logger text file example

6.1.4 MATLAB project

In order to decrease the simulation time on MODELSIM, the VHDL blocks write text files by using binary numbers. While designer controls the text files, it is hard to examine text files which are represented as binary numbers. MATLAB is used to arrange the text files in order to read them easily since MATLAB is faster on writing/reading text files than MODELSIM.

The text file in Fig. 6.4 is the MATLAB output of the text file in Fig.6.2 which is written during the MODELSIM simulation. "SP" represents the source port, "DP" destination port and "BT" birth time which defines the packet's entering time.

cikis_paket_log_FPGA.txt

```
****CYCLE =0
****CYCLE =1
 SP= 0 DP= 1 BT= 0
 SP= 1 DP= 2 BT= 0
 SP= 3 DP= 3 BT= 0
****CYCLE =2
 SP= 2 DP= 2 BT= 0
 SP= 3 DP= 1 BT= 1
****CYCLE =3
 SP= 0 DP= 3 BT= 2
 SP= 1 DP= 2 BT= 1
 SP= 2 DP= 1 BT= 2
****CYCLE =4
 SP= 0 DP= 0 BT= 3
 SP= 1 DP= 3 BT= 2
 SP= 3 DP= 1 BT= 3
****CYCLE =5
 SP= 2 DP= 1 BT= 3
 SP= 3 DP= 3 BT= 2
****CYCLE =6
 SP= 0 DP= 3 BT= 4
 SP= 2 DP= 2 BT= 5
 SP= 3 DP= 0 BT= 4
```

Figure 6.4: Output text file matlab example

The text file in Fig. 6.5 is the MATLAB output of the text file in Fig. 6.3 which is written during MODELSIM simulation as binary .

main_log_FPGA.txt

```
****START OF THE CYCLE =0****
 REQ FROM IN3= 0000 REQ FROM IN2= 0000 REQ FROM IN1= 0000 REQ FROM IN0= 0000
 ACC_POINTER OF IN3= 00 ACC_POINTER OF IN2= 00 ACC_POINTER OF IN1= 00 ACC_POINTER OF IN0= 00
 GRA_POINTER OF OUT3= 00 GRA_POINTER OF OUT2= 00 GRA_POINTER OF OUT1"= 00 GRA_POINTER OF OUT0= 00
****START OF THE CYCLE =1****
 REQ FROM IN3= 0000 REQ FROM IN2= 1000 REQ FROM IN1= 0010 REQ FROM IN0= 0000
 ACC_POINTER OF IN3= 00 ACC_POINTER OF IN2= 00 ACC_POINTER OF IN1= 10 ACC_POINTER OF IN0= 00
 GRA_POINTER OF OUT3= 11 GRA_POINTER OF OUT2= 00 GRA_POINTER OF OUT1"= 10 GRA_POINTER OF OUT0= 00
****START OF THE CYCLE =2****
 REQ FROM IN3= 0010 REQ FROM IN2= 0001 REQ FROM IN1= 0000 REQ FROM IN0= 0000
 ACC_POINTER OF IN3= 10 ACC_POINTER OF IN2= 01 ACC_POINTER OF IN1= 10 ACC_POINTER OF IN0= 00
 GRA_POINTER OF OUT3= 11 GRA_POINTER OF OUT2= 00 GRA_POINTER OF OUT1"= 00 GRA_POINTER OF OUT0= 11
****START OF THE CYCLE =3****
 REQ FROM IN3= 0001 REQ FROM IN2= 0000 REQ FROM IN1= 1000 REQ FROM IN0= 0100
 ACC_POINTER OF IN3= 01 ACC_POINTER OF IN2= 01 ACC_POINTER OF IN1= 00 ACC_POINTER OF IN0= 11
 GRA_POINTER OF OUT3= 10 GRA_POINTER OF OUT2= 01 GRA_POINTER OF OUT1"= 00 GRA_POINTER OF OUT0= 00
```

Figure 6.5: Fabric scheduler logger text file matlab example

Moreover, MATLAB is used in order to calculate packet delay and thrpughput of the overall designs. The text files which is written by MODELSIM includes all necessary information for those calculations.

6.1.5 Design Verification Steps

In the early time of the design, the results of the simulation is controlled manually. The poisson input traffic generator block is used as traffic generator. The RTL simulation is done by MODELSIM. The output packets and internal signals of iSLIP is written as text files. Those text files are processed by MATLAB and then, the functionality of the design is controlled manually. The 4x4 basic iSLIP, 4x4 2-LSP IMC VOQ iSLIP and 4x4 2-LLP IMC VOQ iSLIP are verified in this way.

It is hard to verify the design manually for high numbers of time-slots. In order to do that for long period, the C++ based iSLIP simulator is employed [41]. The same input traffic text file is used with the C++ based iSLIP simulator. The traffic reader block is used during MODELSIM simulation to apply input traffic from the text file. After the simulation is completed on MODELSIM, the text files including output packet traffic are automatically compared by using MATLAB. The processes are applied for the 4x4 basic iSLIP, 4x4 2-LSP IMC VOQ iSLIP and 4x4 2-LLP IMC VOQ iSLIP. At the end, the overall designs which are implemented by using VHDL and C++ verify each other.

6.2 Simulations

Performance metrics are explained in the chapter 2. The router performance is evaluated in terms of average delay and throughput at different traffic loads.

The 4x4 basic iSLIP and the 4x4 2-LLP IMC VOQ iSLIP are compared. Average packet delays and throughput metrics are examined at different load values under Poisson traffic. The half of the incoming packets consists of high priority class packets, the other half of it consists of low priority class packets. Furthermore, the packet size distribution is fixed, all incoming packets have a length of one fixed-size cell.

The rate of the incoming packets divided by the line rate is defined as offered load.

Average packet delay values are given in terms of time slot values. In the simulations, a packet which comes in time slot t can exit from the router at the time-slot $t + 1$ earliest. Thus, all packets have a fixed one time slot delay because of the processing. Note that, one time slot processing delay is subtracted in order to calculate queueing delay per packet. The average delay per packet values are calculated as queueing delay in [7]. Average packet delay values are calculated for packet delay average of all the output ports.

In the simulations of 2-LLP IMC VOQ iSLIP, the limit value which is defined as *MaxSuccesiveAllowedRequests* in the algorithm 2 equals to 4.

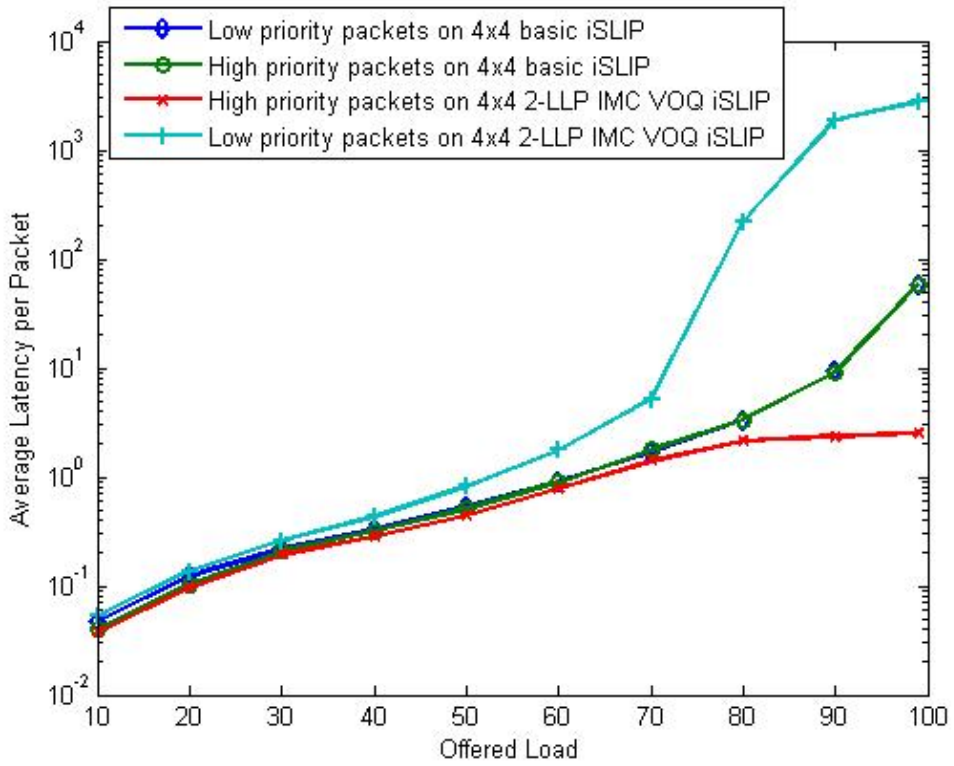

Figure 6.6: Average queueing delay for low priority class packets and high priority class packets in 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP under Poisson traffic

In the Fig.6.6, average queueing delay for low priority class packets and high priority class packets in the routers of the 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP are compared. The average queueing delay of high priority class and low priority class

packets are almost the same in the 4x4 basic iSLIP as it is expected since the basic iSLIP does not provide QoS differentiation. On the other hand, especially for 80% and higher offered load values, the high priority class packets have lower average queuing delay. While supplying lower average delay for the high priority class packets, the low priority class packets face with higher delay values than delay values in the basic iSLIP. To this end, it is figured out that the 2-LLP IMC VOQ structure makes average queueing delay per packet lower for the high priority class packets.

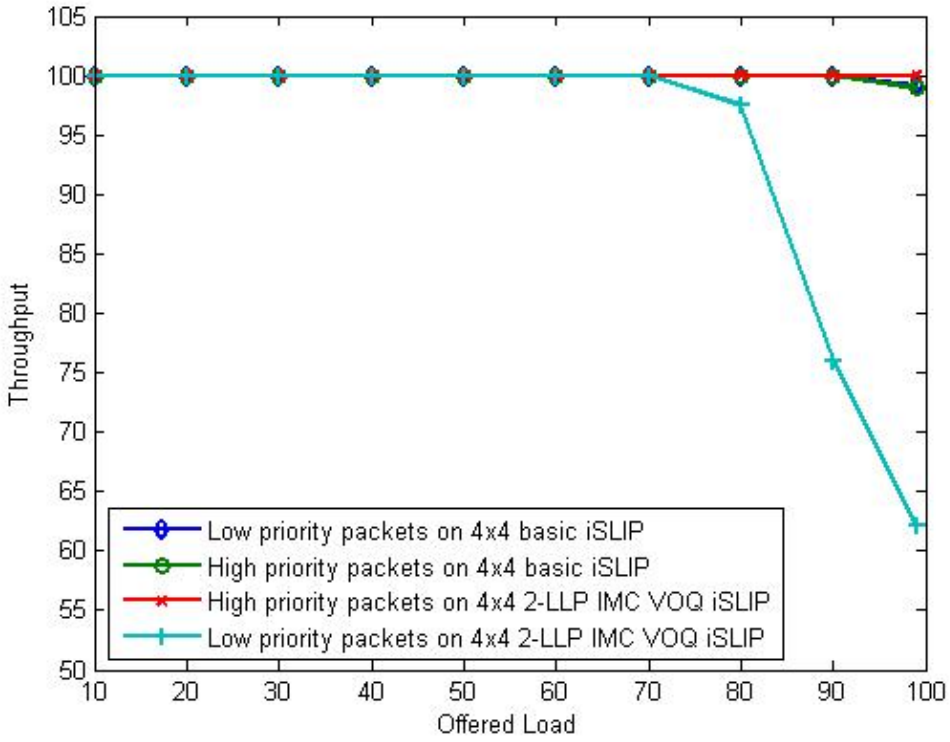

Figure 6.7: Throughput for low priority class packets and high priority class packets in 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP under Poisson traffic

Throughput is calculated as the number of total incoming packets are divided by the number of total outgoing packets, then multiplied by 100 in order to get the percentage.

In [7], it is stated that the iSLIP can achieve 100% throughput under poisson traffic. In the Fig.6.7, the 4x4 basic iSLIP achieves 100% throughput for both low priority class packets and high priority class packets for offered load values below 90%. When the offered load reaches 99%, the 4x4 basic iSLIP achieves almost 100% throughput.

On the other hand, although 4x4 2-LLP IMC VOQ can achieve 100% throughput for high priority class packets for all offered load values, throughput for low priority class packets considerably worse especially on the offered load values below 80%.

Extensive simulations under different scenarios can be found in [41].

6.3 FPGA resource usage evaluation

After completing verification of the VHDL codes, the VHDL codes are synthesized by Xilinx ISE 12.4. The target FPGA is selected as Virtex-5 XC5VFX130T which is a product of Xilinx.

The 4x4 basic iSLIP, 4x4 2-LLP IMC VOQ and 8x8 2-LLP IMC VOQ are compared in terms of resource usage. 2-LCP IMC VOQ iSLIP is not included since the resource usage of the 2-LCP IMC VOQ iSLIP is almost the same as the 2-LLP IMC VOQ iSLIP.

In the chapter 5, the changing the number of the priority levels of IMC VOQ iSLIP is explained. Moreover, the basic iSLIP can be designed by changing the number of priority levels to 1. To this end, the 4x4 basic iSLIP VHDL codes are generated by applying the same procedure which is presented in the chapter 5. Moreover, in the chapter 5 it is stated that the resource usage of a single input line block scales with $O(P)$ while changing the number of priority levels. The number of priority levels equals to 1 for the basic iSLIP and 2 for the 4x4 IMC VOQ iSLIP. Thus, the resource usage of single input line block in the 4x4 2-LLP IMC VOQ iSLIP is almost twice of the single input line block in 4x4 basic iSLIP as it can be seen from table 6.1.

Table 6.1: Resources used in single input line block

	4x4 basic iSLIP	4x4 2-LLP IMC VOQ iSLIP
Number of Slice Registers	631	1125
Number of Slice LUTs	430	836
Number of Block RAM/FIFO	4	8

In the table 6.2, the resource usage of the complete 4x4 basic iSLIP and the 4x4

2-LLP IMC VOQ iSLIP as explained in the chapter 4 is compared. The difference between the two designs are completely resulted from the changes in the input line blocks as it is stated in the chapter 5. The two designs are completely the same except the input line blocks. Each design consists of four input line blocks. Thus, the values in the table 6.1 must be multiplied by 4. Then, the result of multiplication must be extracted from the values in the table 6.2 in order to acquire the resource usage of the rest of the designs. The values are almost the same which shows that the rest of the designs is almost the same for the 4x4 2-LLP IMC VOQ iSLIP and the 4x4 basic iSLIP.

The maximum clock frequency in the table 6.2 that is calculated by Xilinx ISE 12.4 indicates the design's maximum operation frequency. The maximum clock frequency in the 4x4 basic iSLIP and the 4x4 2-LLP IMC VOQ iSLIP are completely the same. The maximum clock frequency is probably limited by a region which is outside the input line block. The areas which is outside the input line blocks are the same; therefore, the maximum clock frequency is the same for both of the two designs. Furthermore, by looking at the maximum clock frequency, the capacity of the router can be roughly calculated. The data width equals to 32 and the maximum clock frequency is accepted as 300 MHz. That is, each input port can transmit at 9.6 Gbps (32x300 Mbps). The router capacity equals to 38.4 Gbps (4x9.6 Gbps) since there are 4 input ports.

Table 6.2: Resources used in 4x4 basic iSLIP and 4x4 2-LLP IMC VOQ iSLIP

	4x4 basic iSLIP	4x4 2-LLP IMC VOQ iSLIP
Number of Slice Registers	3489	5461
Number of Slice LUTs	2703	4319
Number of Block RAM/FIFO	20	36
Maximum Clock Frequency	327.332MHz	327.332MHz

Moreover, the number of ports in the 4x4 2-LLP IMC VOQ iSLIP is increased to 8x8 as it is explained in the chapter 5. The resources used in the 4x4 2-LLP IMC VOQ iSLIP and the 8x8 2-LLP IMC VOQ iSLIP are compared in the table 6.3.

The number of block RAMs is consistent with the expectations. In the 4x4 2-LLP IMC VOQ iSLIP, there are 4 input line blocks. Each input line block uses 8 block

RAMs. Thus, the total 32 block RAMs are used by the input line blocks in the 4x4 2-LLP IMC VOQ iSLIP. Moreover, there are 4 output line blocks. Each output line block uses 1 block RAM. So far as this is the situation, the total block RAM usage of the output line blocks is equal to 4. To this end, 36 block RAMs are used in the 4x4 2-LLP IMC VOQ iSLIP, 32 of them are used by the input line blocks and 4 of them are used by the output line blocks. On the other hand, there are 8 input line blocks and 8 output line blocks in the 8x8 2-LLP IMC VOQ iSLIP. Thus, the total 136 block RAMs are used, 128 of them are used by the input line blocks and 8 of them are used by the output line blocks.

Moreover, while changing the number of ports, the resources are expected to scale with $O(N^2)$. Hence, the resource usage of the 8x8 2-LLP IMC VOQ iSLIP is almost 4 times of the 4x4 2-LLP IMC VOQ iSLIP.

Additionally, the maximum clock frequency of the 8x8 2-LLP IMC VOQ iSLIP is lower than the maximum clock frequency of the 4x4 IMC VOQ iSLIP. While increasing the port number, the length between the input ports and the output ports rises in the hardware design. The increase in length results in a decrease in the maximum clock frequency. Furthermore, when maximum clock frequency is accepted as 200 MHz for the 8x8 one, the input line capacity at 6.4 Gbps (32x200 Mbps) is supported. Therefore, the router capacity equals to 51.2 Gbps (8x6.4 Gbps).

Table 6.3: Resources used in 4x4 2-LLP IMC VOQ iSLIP vs 8x8 2-LLP IMC VOQ iSLIP

	4x4 2-LLP IMC VOQ iSLIP	8x8 2-LLP IMC VOQ iSLIP
Number of Slice Registers	5461	18517
Number of Slice LUTs	4319	15334
Number of Block RAM/FIFO	36	136
Maximum Clock Frequency	327.332MHz	213.038MHz

6.4 Results and Discussion

By examining the simulation results, it is realized that the 2-LLP IMC VOQ structure make average queueing delay for higher priority packets lower especially for the high

load values above 80%. On the other hand, the 2-LLP IMC VOQ structure results in a higher average queueing delay and a lower throughput values for the lower priority packets for especially high load values above 80%.

In fact, getting worse in throughput and delay values for lower priority packets are expected. On the other hand, $MaxSuccesiveAllowedRequests$ in algorithm 2 can be changed in order to achieve more acceptable results for lower priority packets. Moreover, different algorithms can be developed to acquire more acceptable throughput and delay values for lower priority packets. Our 2-LLP IMC VOQ algorithm is based on the combined selection approach which is explained in the Section 3.2. However, different algorithms based on the separated selection approach may be developed despite that its implementation complexity is higher.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Today networks support wide range of applications which have different QoS requirements. Thus, it is important to provide different levels of QoS for different applications.

The supporting QoS at network layer devices has been an interesting research topic with the increasing data rate. Although many QoS scheduling algorithms are proposed in the literature for output queued routers, they require internal speedup which increases their implementation cost. On the other hand, the input queued routers can achieve acceptable QoS without internal speedup. However, the delay at the input queues degrades the QoS. In the input queued routers, the fabric scheduler component which is responsible for input-output matching decision directly affects the queueing delay. Thus, it is possible to supply better QoS by improving the fabric schedulers.

This thesis presents a per-flow queue structure to replace the standard VOQs in the input queued routers together with a request preprocessing unit to support QoS at the network layer. The proposed IMC VOQs can be integrated with a generic fabric scheduler without changing its operation to achieve service differentiation. The QoS characteristic of the fabric scheduler can be changed by the request selection algorithm of the IMC VOQs since the requests which are sent to fabric scheduler are arranged by the IMC unit in each input port.

The IMC VOQs is realized on FPGA together with the well known iSLIP as the generic fabric scheduler. Furthermore, FPGA implementation supports variable size packets with the help of segmentation and reassembly modules. Moreover, switch

fabric design is also implemented in order to transfer packets from input ports to output ports. The overall architecture is evaluated with respect to both its capability for QoS support and its hardware resource consumption.

The simulation results show that lower average delay per packet is achieved for high priority class packets by integrating IMC VOQs which adopts the proposed request selection algorithm. On the other hand, throughput and average delay is getting worse for low priority class packets when IMC VOQs is implemented. Furthermore, the resource usage values are consistent with our expectations. That is, the resource usage can be roughly calculated for large number of ports and different numbers of priority levels without completing the overall design.

The modularity of the design is important; therefore, FPGA devices are suitable platforms for the router design. The FPGA implementation which is provided in this thesis can not be used instead of a router in the network since it handles only packet headers. The packet payloads must be also supported together with packet headers. Moreover, in the practice of the commercial routers, the input line blocks, output line blocks, switch fabric and fabric scheduler components are realized in different FPGAs which are located on different Printed Circuit Boards(PCB). In this thesis, all those blocks are implemented in the same FPGA as an example. Thus, each block must be implemented on different PCBs including FPGA component. Furthermore, different FPGA components are selected for each block according to the block requirements. For instance, the FPGA component which is used in input line cards should have large storage capacity in order to realize the VOQs. On the other hand, the FPGA component which is used as switch fabric should support large bandwidth capacity in order to transfer packets from input line cards to output line cards.

The future work includes incorporating a similar intelligent unit at the output to achieve further control on the fabric scheduler decision to support QoS. Different IMC request selection algorithms will also be investigated. Moreover, IMC VOQs can be integrated with the different fabric scheduling algorithms in order to observe the effects of IMC VOQs on them. Furthermore, the proposed FPGA architecture will be adapted to develop and implement new fabric schedulers which can exchange different types of information between inputs and outputs such as queue states. The

FPGA implementation will be improved in order to handle with packet payloads together with packet headers. Lastly, real-time experiments can be realized with FPGA implementation in addition to MODELSIM simulations.

REFERENCES

[1] [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5548p-switch/white_paper_c11-622479.html [Accessed: 23/8/2014]

[2] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100throughput in an input-queued switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, Aug 1999.

[3] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *Communications, IEEE Transactions on*, vol. 35, no. 12, pp. 1347–1356, Dec 1987.

[4] F. N. E. Leonardi, M. Mellia and M. A. Marsan, "Bounds on the delay and queue lengths in input-queued cell switches," *Journal of the ACM*, vol. 50, no. 12, pp. 520–550, Jul 2003.

[5] Y. Shen, S. Panwar, and H. Chao, "Design and performance analysis of a practical load-balanced switch," *Communications, IEEE Transactions on*, vol. 57, no. 8, pp. 2420–2429, Aug 2009.

[6] [Online]. Available: http://www.cisco.com/c/en/us/td/docs/routers/crs/crs1/16_slot_lc/system_description/reference/guide/qsysdsc/sysdsc4.html [Accessed: 23/8/2014]

[7] N. McKeown, "The islip scheduling algorithm for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 2, pp. 188–201, Apr 1999.

[8] A. E. Kateeb, "High-speed routers design using data stream distributor unit," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 133 – 144, 2007.

[9] "Optimizing Video Transport in Your IP Triple Play Network," Cisco Systems Inc, Tech. Rep., 2006.

[10] T. Szymanski and D. Gilbert, "Internet multicasting of iptv with essentially-zero delay jitter," *Broadcasting, IEEE Transactions on*, vol. 55, no. 1, pp. 20–30, March 2009.

[11] V. Murcia, M. Delgado, T. Vargas, J. Guerri, and J. Antich, "Vaipa: A video-aware internet protocol architecture," in *High Performance Switching and Rout-*

ing (HPSR), 2011 IEEE 12th International Conference on, July 2011, pp. 140–145.

[12] R. Luebben, G. Li, D. Wang, R. Doverspike, and X. Fu, "Fast rerouting for ip multicast in managed iptv networks," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*, July 2009, pp. 1–5.

[13] "Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks," *Journal of Network and Computer Applications*, vol. 32, no. 4, pp. 901 – 912, 2009.

[14] G. Muntean, P. Perry, and L. Murphy, "A new adaptive multimedia streaming system for all-ip multi-service networks," *Broadcasting, IEEE Transactions on*, vol. 50, no. 1, pp. 1–10, March 2004.

[15] Y. Huang, R. Guerin, and P. Gupta, "Supporting excess real-time traffic with active drop queue," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 5, pp. 965–977, Oct 2006.

[16] Y. Bai and M. Ito, "Application-aware buffer management: new metrics and techniques," *Broadcasting, IEEE Transactions on*, vol. 51, no. 1, pp. 114 – 121, mar. 2005.

[17] G. Nong and M. Hamdi, "On the provision of quality-of-service guarantees for input queued switches," *Communications Magazine, IEEE*, vol. 38, no. 12, pp. 62–69, Dec 2000.

[18] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *Communications, IEEE Transactions on*, vol. 35, no. 12, pp. 1347–1356, Dec 1987.

[19] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 3, pp. 344–357, Jun 1993.

[20] A. Parekh and R. Gallagerr, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *Networking, IEEE/ACM Transactions on*, vol. 2, no. 2, pp. 137–150, Apr 1994.

[21] T. Szymanski, "A low-jitter guaranteed-rate scheduling algorithm for packet-switched ip routers," *Communications, IEEE Transactions on*, vol. 57, no. 11, pp. 3446–3459, Nov 2009.

[22] D. Pan and Y. Yang, "Bandwidth guaranteed multicast scheduling for virtual output queued packet switches," *Journal of Parallel and Distributed Computing*, vol. 69, no. 12, pp. 939–949, Dec. 2009.

[23] [Online]. Available: http://www.cisco.com/c/en/us/products/video/enterprise-content-delivery-system-ecds/index.html [Accessed: 23/8/2014]

[24] [Online]. Available: http://www.cisco.com/c/en/us/products/interfaces-modules/shared-port-adapters-spa-interface-processors/index.html [Accessed: 23/8/2014]

[25] H. J. Chao and B. Liu, *High Performance Switches and Routers*. Wiley, 2007.

[26] S. Atalla, D. Cuda, P. Giaccone, and M. Pretti, "Belief-propagation-assisted scheduling in input-queued switches," *Computers, IEEE Transactions on*, vol. 62, no. 10, pp. 2101–2107, Oct 2013.

[27] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 6, pp. 1030–1039, Jun 1999.

[28] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-mode scheduling in input-queued cell-based switches," *Networking, IEEE/ACM Transactions on*, vol. 10, no. 5, pp. 666–678, Oct 2002.

[29] B. Zhang, X. Wan, L. Junzhou, and X. Shen, "A nearly optimal packet scheduling algorithm for input queued switches with deadline guarantees," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[30] Y. Lee, J. Lou, J. Luo, and X. Shen, "An efficient packet scheduling algorithm with deadline guarantees for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 15, no. 1, pp. 212–225, Feb 2007.

[31] J. E. Hopcroft and R. M. Karp, "An algorithm for maximum matchings in bipartite graphs," *Soc. Ind. Appl. Math J. Computation*, vol. 2, pp. 225–231, 1973.

[32] I. Keslassy, R. Zhang-Shen, and N. McKeown, "Maximum size matching is unstable for any packet switch," *Communications Letters, IEEE*, vol. 7, no. 10, pp. 496–498, Oct 2003.

[33] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, Nov. 1993.

[34] J. Chao, "Saturn: a terabit packet switch using dual round robin," *Communications Magazine, IEEE*, vol. 38, no. 12, pp. 78–84, Dec 2000.

[35] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Tech. Rep., 1995.

[36] P. Bommannavar, J. Apostolopoulos, and N. Bambos, "Deadline aware packet scheduling in switches for multimedia streaming applications," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 3797–3802.

[37] K. F. Chen, E. H.-M. Sha, and S. Zheng, “Fast and noniterative scheduling in input-queued switches: Supporting QoS,” *Computer Communications*, vol. 32, no. 5, pp. 834–846, Mar. 2009.

[38] B. Prabhakar and N. McKeown, “On the speedup required for combined input- and output-queued switching,” *Automatica*, vol. 35, no. 12, Dec. 1999.

[39] A. Bianco, P. Giaccone, G. Masera, and M. Ricca, “Power control for crossbar-based input-queued switches,” *Computers, IEEE Transactions on*, vol. 62, no. 1, pp. 74–82, Jan 2013.

[40] [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator_ug175.pdf [Accessed: 23/8/2014]

[41] A. Ada, “Implementation and performance analysis of switch fabric schedulers with a new accurate simulator software,” Master’s thesis, Middle East Techncal University, Ankara, Turkey, 2014.

[42] [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/platform_studio/ps_c_sim_behavioral_simulation.htm [Accessed: 23/8/2014]