Design and Implementation of Load-Balanced Multipath Self-routing Switching System

Qian Zhan, Hui Li*, Fuxing Chen, Li Ma

Abstract—**In order to ensure high quality of service (QoS) for Next Generation Network (NGN), we construct an innovative Load-Balanced Multipath Self-routing Switching Structure which consists of the same two multipath self-routing fabrics. The result of simulation is inspiring for achieving 100% throughput and no delay or jitter. For this reason, we start on the implementation on an Altera StratixIV FPGA. And the whole FPGA system is designed into two collaborative components: the UDP system and the register system. With two algorithms around input and output two stages, incoming traffic is transformed into uniformity and then to their final destinations. During the later period debugging, software simulation platform and automated test platform are built, which contribute to our work very much. At last, we carry out several experiments to test and verify our system. The report of the test result accords with what we expected.**

Index Terms—**Next Generation Network, Load-Balanced, Self-routing, Switching Fabric, FPGA**

I. INTRODUCTION

In recent years, with a rapid increase in the number of Internet users, the network scale expands unceasingly. Rich Internet applications, especially the popularity of online video services [1], contribute to the network congestion that almost everyone experienced. This phenomenon puts forward a huge challenge to the vital component, the router. Actually, the router has become a significant bottleneck in the development of the network. On the other hand, on the basis of TCP/IP, the network layer of Internet only provides the best effort delivery rather than the commitment for quality of service (QoS) [2].

In order to improve the performance of routers and reduce implementation costs, various kinds of solutions are proposed. The Load-Balanced Birkhoff-von Neumann switch [3] interests us for that it can achieve 100% throughput under most network traffic by using a balancer to equalize input flows. However, the structure does not maintain the order of packets after switching and the average packet queuing delay increases linearly with the number of ports. Obviously, it is not suitable for large scale extension. On the contrary, another structure we focus on, the Banyan-based Quasi-Circuit Switch [4] has low component complexity O ($N\log_2 N$, N is the number of ports) and the ability of self-routing and distributed processing. However, because of the blocking feature, QoS is not ensured.

Based on the advantages and disadvantages of the above two kinds of structures, we propose a Load-Balanced Multipath Self-routing Switching Structure by connecting two multipath self-routing fabrics in series. The first one acts as a balancer and the other one severs as a router. Concentrators, which are made up by basic sorting units, are sorted by the arrangement rules of Multistage Interconnection Network to construct the whole structure. Theoretical analysis and NS2 simulation indicate that our model can obtain 100% throughput under normal circumstances and easy to be expanded in size [5].

Further, we translate the theoretical model into a modular FPGA system which consists of two main parts: the UDP system and the register system. And then, the whole system has been implemented on an Altera StratixIV FPGA. In the testing phase, our system works steadily and efficiently and meets the basic requirements for QoS applications.

The rest of the paper is organized as follows. Theoretical basis and modeling are introduced in Section II. Section III describes the system design and implementation based on FPGA. Section IV presents system testing with real network traffic, and then Section V summarizes the whole work.

II. THEORETICAL BASIS AND MODELING

A. 2×2 Basic Sorting Unit

The 2×2 basic sorting unit is a sequential logic circuit, with two inputs and two outputs (respectively called 0/1 port). According to the theory of algebraic distributive lattices [6], we define the two inputs as Ω_0 and Ω_1, each of which has three kinds of data: the one going to output0, the one going to output1 and the invalid data. As list in Table I, the sorting unit has two essential states: Cross and Bar. That means the inputs go to the different outputs: input0/input1 to output1/output0 and input0/input1 to output0/output1, corresponding to Cross and Bar, respectively.

Manuscript received December 23, 2014; revised January 10, 2014. This work was supported in part by the National Basic Research Program of China (973 Program) under Grant 2012CB315904, the National Natural Science Foundation of China under Grant 61179028, the Natural Science Foundation of Guangdong Province under Grant 201101000923 and 2013020012822, the Basic Research of Shenzhen under Grant 201104210120A and 20130331144502026.

Qian Zhan is with the Peking University Shenzhen Graduate School, Shenzhen, GuangDong 518055 P.R.China (e-mail: zhanqian0218@ gmail.com).

Hui Li is with the Peking University Shenzhen Graduate School, Shenzhen, GuangDong 518055 P.R.China. He is the director of Shenzhen Eng. Lab of Converged Networks Technology and the deputy director of Shenzhen Key Lab of Cloud Computing Tech. & App. (phone: 0086-755-26035354; mobile phone: 0086-13602672514; e-mail: huilihuge@163.com).

Fuxing Chen is with the Peking University Shenzhen Graduate School, Shenzhen, GuangDong 518055 P.R.China (e-mail: chenfuxing@ pku.edu.cn).

Li Ma is with the Peking University Shenzhen Graduate School, Shenzhen, GuangDong 518055 P.R.China (e-mail: mali5057@163 .com).

TABLE I. THE STATES OF 2×2 SORTING UNIT

States		Ω_1		
		0-bound	***idle***	***1-bound***
Ω_0	***0-bound***	CONF	BAR	BAR
	idle	CROSS	EITHER	BAR
	1-bound	CROSS	CROSS	CONF

If the inputs compete for the same one output, the state will be Conflict and the final choice of BAR or CROSS will depend on their priority. All the three states are clearly shown in Fig. 1.

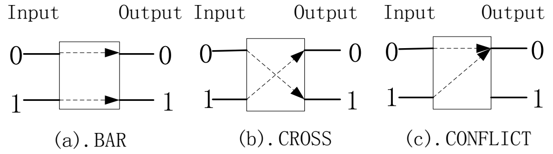


Fig. 1 basic sorting unit and its states

B. *Inter-stage Bit-permuting Model*

An N×N (N=2^n) self-routing network is a Multistage Interconnection Network (MIN) built by 2×2 basic sorting units. By using first stage permutation σ_0, inter-stage permutation σ_1, σ_2 … $\sigma_{(n-1)}$ and last stage permutation σ_n, the network can be represented as [σ_0: σ_1: σ_2:…: $\sigma_{(n-1)}$: σ_n]. Each colon symbolizes a stage of 2x2 units. We can define a Trace sequence and a Guide sequence [8] as follows:

$$T_k=(\sigma_0\sigma_1\ldots\sigma_{K-1})^{(-1)}(n) \quad 1\le k\le n \qquad (1)$$

$$G_k=(\sigma_0\sigma_1\ldots\sigma_{K-1})(n) \quad 1\le k\le n \qquad (2)$$

Route is specified by Trace or Guide. As Fig. 2 shows, for the network [: (43): (42) (31): (43):], data from the origination address $I_1I_2I_3I_4$ finally gets to the destination $O_1O_2O_3O_4$ with the decision at each stage by the Trace or Guide.

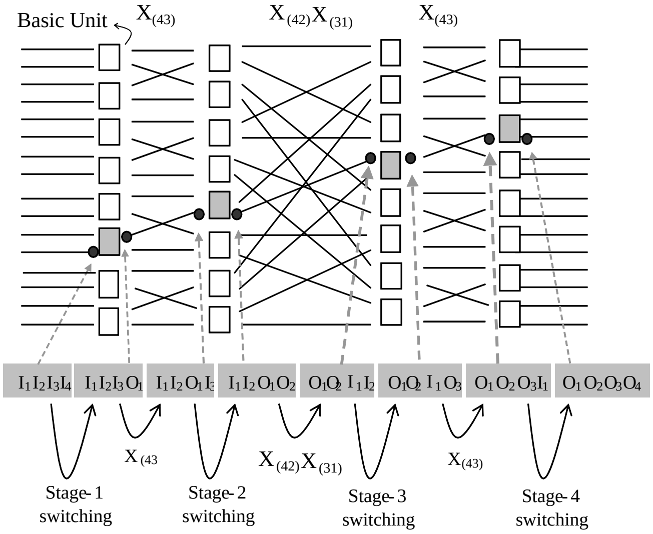


Fig. 2 an example of self-routing network

C. *Multipath Self-routing Switching Structure*

Multipath Self-routing Switching Structure (MSSS) [9] is an innovative structure, which combines MIN with concentrators.

To construct MSSS, we substitute each basic sorting unit for the 2G-to-G concentrator and replace the single cable with a bundle of G cables. Fig. 3 illustrates the multipath structure (N=128 M=16 and G=8) which is based on a 16×16 self-routing network. G shows the size of group, M is the number of group and N=M×G indicates the whole number of input/output ports (G=$2^{g,}$ M=2^m, N=2^n, n=m+g, n, m, g are positive integers). Acting as an indispensable part of MSSS, the 2G-to-G concentrator [10] separates the larger G signals of the whole 2G inputs from the other G signals, finally forming two ordered output groups.

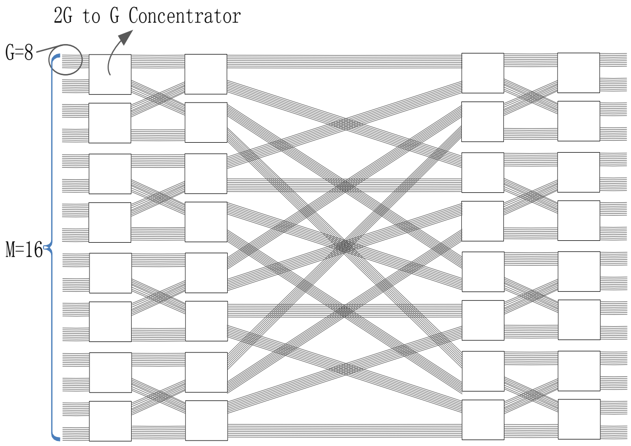


Fig. 3 Multipath Self-routing Switching Structure (M=16, G=8)

D. *Load-Balanced Multipath Self-routing Switching Structure*

As shown in Fig. 4, two MSSSs are used in series to compose the Load-Balanced Multipath Self-routing Switching Structure (Load-Balanced MSSS), with the VOGQs (Virtual Output Group Queues) [5] ahead of the first fabric and the assemblages at the end of the second fabric. Actually, by using simple algorithms and small buffers, the first stage fabric serves as a balancer, which spreads the incoming traffic to all the ingress ports of the second stage fabric. Then the second stage fabric forwards the data in a self-routing manner to their final destinations. Every G inputs/outputs are bundled into an input/output group, thus N input lines form M groups on the input side (N=M×G), so is the output side. To ease presentation, IG/OG denotes input/output group, and MG represents a line group between the two stages. In the project, there are 4 IGs, 4 MGs and 4 OGs. Each group has 8 lines.

VOGQs are responsible for storing packets and making data ready for IGs. We use VOGQ (i,j) to denote the VOGQ whose packets are destined for OGj from IGi.

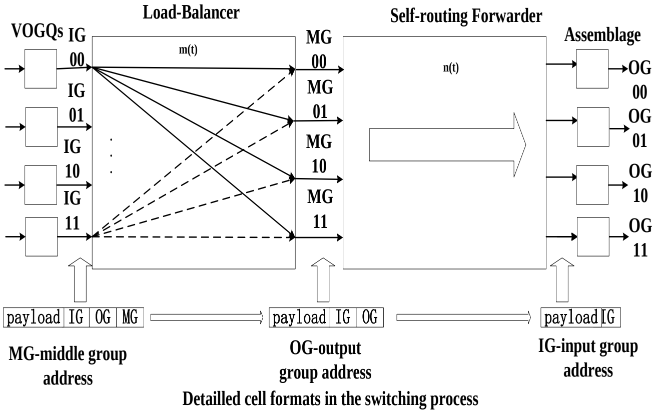


Fig. 4 Load-Balance Multipath Self-routing Switching Structure

Generally, for the structure we proposed, the processing of arriving packets in each time slot is composed by several sequential phases which are shown as follows. In addition, to

ISBN: 978-988-19253-3-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

achieve maximum processing speed, we should use pipeline structure as far as possible.

1) *Preparatory phase*: With checking and judging, the arriving packet which is destined for OGj from IGi is stored into VOGQ (i,j).
2) *Splitting phase*: Packets in VOGQs are split into cells according to Algorithm 1. And each cell will be added with some certain packet headers.
3) *Balancing phase*: With the help of MG tags, cells will be routed to every middle group simultaneously and uniformly. When the cells reach the middle groups, MG tags will be dropped.
4) *Routing phase*: Cells are further to their final destinations directed by OG tags. When they get through the second stage fabric, OG tags will be discarded.
5) *Assembling phase*: Cells are to be assembled to original packets according to Algorithm 2. When completed, packets will be output from the OGs.

Algorithm 1: For each input group, packets stored in VOGQs should be split into cells with equal length during splitting phase. Furthermore, we add MG tags, OG tags, IG tags and some other control messages ahead of each cell. The MG tags are set artificially. For example, a packet is split into five cells and their respective MG tags should be 0, 1, 2, 3, 0, orderly. If the following packet can be split into three cells, the tags will be 1, 2 and 3. The rule is also suitable for other various packets.

Algorithm 2: For each output group, cells with the same IG address are assembled during assembling phase. IG tags, sequence numbers and flags of the last cells will help us to reorganize the scrambled cells. For example, we get a few of cells in OG address 01. Some of them have the same IG tag 00 and the sequence numbers 3,2,4,1. By the way, the cell with sequence number 4 is marked with the trailing flag. The others own IG tag 10 and the sequence numbers 3, 2, 1, but no cell has the trailing flag. By now, we can easily get the packet which is from IG 00 by connecting the cells together in the order 1,2,3,4. For the packet from IG 10, we still need to wait for the last cell to arrive.

III. SYSTEM DESIGN AND IMPLEMENTATION BASED ON FPGA

We use Verilog HDL to carry on the main design and Tcl script language to build an operating platform for the register system. Functional simulation is also an important part, which is implemented by Perl and Makefile.

A. The Overall Architecture of the System

The whole system is implemented on an Altera StratixIV FPGA, with a Marvell 88E1111 PHY chip being used for physical layer. The TSE (Triple-Speed Ethernet) IP core, interacted with the PHY chip through RGMII interface, provides standard Ethernet frames.

We divide the system into two main parts: the user data path (UDP) system (There are the same four UDP systems in the whole system, each of which servers a group of MSSS and we just need to introduce one of them in the paper.) and the register system. They are independent structurally and interrelated functionally. The UDP system is responsible for data processing and cell switching with many sub-modules, FSMs (Finite-State Machines) and FIFOs in it. In order to facilitate debugging, we have designed the register system to monitor signals and states in UDP in real-time.

Lastly, the logic utilization is 32% according to the compilation report generated by Quartus II 11.0.

B. Design and Analysis of User Data Path System

Packets enter into the system through the RJ45 network port firstly. And after being processed in physical layer by PHY chip, they will be sent to the UDP system, which is the major part of data processing. There are four main functions in UDP. First of all, we can extract necessary information from packets or cells, such as the packet length, priority and the target address, etc. Second, by utilizing the information we extract, the UDP system generates various packet headers, which will be very useful to assist the data processing. Third, it achieves load balancing and self routing by constructing the switching fabric. At last, it completes the assembling of the cells.

The left part of Fig. 5 gives us a full view of the process. The solid arrows indicate the direction of data flow. We can see that input packets pass through nine sub-modules (not including PHY) in turn and get back to PHY.

The functions of each sub-module are as follows.

1) *Sgmii_ethernet*: It is an interface module between the UDP system and the external PHY chip. Mainly constructed by Altera Triple-Speed Ethernet (TSE) IP cores, it provides standard Ethernet frames.
2) *Rx_queue*: This sub-module accepts frames, extracts length information and generates the splitting header. The information is important for Splitter and will be kept until Assemblage.
3) *Output_lookup*: It extracts the information of destination address and priority, which form the lookup header for Router.
4) *Splitter*: For efficiency and easiness of implementation, the following sub-modules are designed based on cells. So, the Splitter will be a key sub-module. It splits each packet into several cells and generates three kinds of headers, the load balancing header for Balancer, the self-routing header for Router and the assembling header for Assemblage.
5) *Arbiter*: As we know, the group size of MSSS we proposed is G=8. Thus, cells should be placed as a group of eight lines. The size of data on each line is 10bit (8 bits for payload and 2 bits for a control signal). This is what Arbiter do.
6) *Balancer*: The structure is the same as MSSS. It transforms the incoming traffic into uniformity with the help of the load balancing header.
7) *Router*: It is also a MSSS. Cells switch here and then go to their final destinations. The process is directed by the self-routing header.
8) *Assemblage*: After switching, groups of cells arrive at every time slot. Assemblage assembles them back to standard Ethernet frames and generates the beginning mark header, which is just to show the very starting of each frame.
9) *Tx_queue*: It contains some memory buffers to cache the

frames and then sends them back to Sgmii_ethernet.

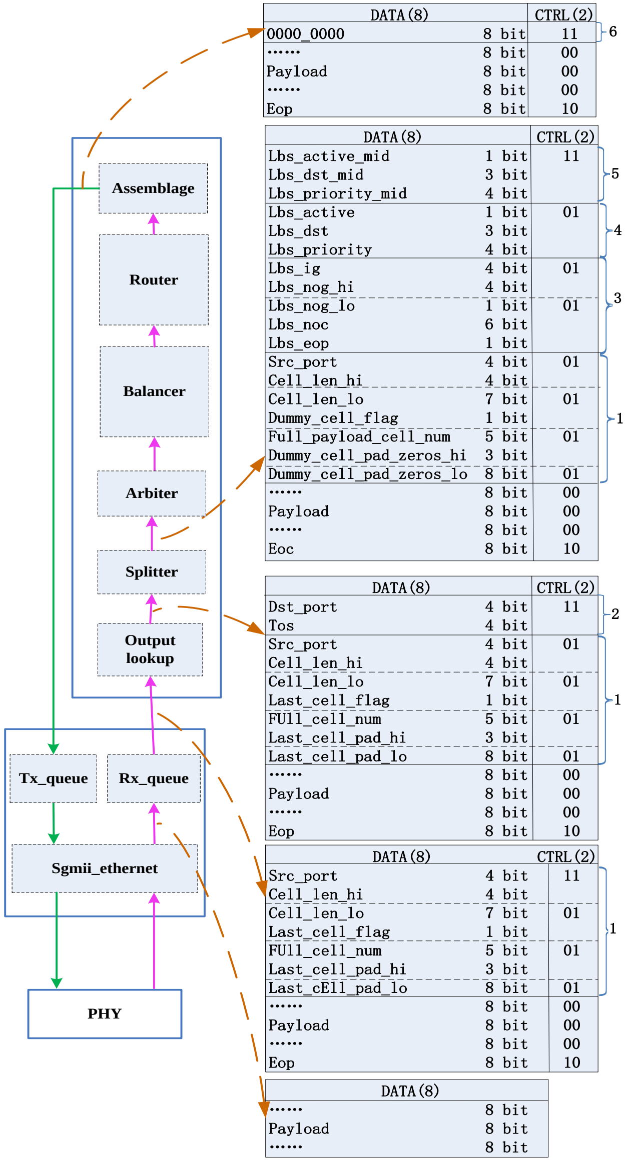


Fig. 5 data processing and data format in UDP

The right part of Fig. 5 pointed by dotted arrows describes the specific data format in each sub-module. Moreover, as shown in the right part, the standard size of data in UDP is 8bit. After going across Rx_queue, a packet header is produced along with the extra added CTRL signal. The 2bit CTRL signal will be transmitted with data in parallel to assist data identification and processing. The signal “11” signifies that the concurrent data is the first byte of a fresh new packet or cell. And “01” represents the various kinds of headers we generate. “00”shows the payload data and “10” tells us the end of a packet or cell.

To sum up, the UDP system extracts the information from incoming frames, and then generates six kinds of headers which are attached in front of the former frames. We use labels “1”, “2”, “3”, “4”, “5”, “6” to represent splitting header, output header, assembling header, self-routing header, load balancing header and beginning mark header, respectively.

1) *Splitting header*: It is created by Rx_queue and contains 4 bytes information. Src_port is the source address of the current frame. The 4bit signal can represent 16 ports, but the group size of our system is M=4. Obviously, it is convenient for scale expansion in the future. The sum of cell_len_hi and cell_len_lo is 11bit, which indicates the standard length of the cell. We set them 2`b0001 and 7`b000_0000, meaning 128Byte. Last_cell_flag indicates whether the current frame can be split into a certain number of cells exactly. Set to high, the signal means that the size of the packet isn`t an integral multiple of 128Byte. Thus the last cell should be padded some bytes to keep the same length. Then, last_cell_pad_hi and last_cell_pad_lo show the number of bytes to be padded. Finally, signal full_cell_num is the number of complete cells.
2) *Output header*: Compared with others, it`s a simple one for containing only one byte information. Dst_port is the destination port of the packet and tos means the priority.
3) *Assembling header*: Generated by Splitter and carrying important information for Assemblage, this header helps to assemble the cells later. Lbs_ig corresponds to the input group number IG shown in Fig. 4. We know that Arbiter places the cells on a group of eight lines. Lbs_nog_hi and lbs_nog_lo point out which line the cell belongs to. We can enlarge the group size to 32 furthest. Lbs_noc is the significant serial number, declaring the actual position of the cell in the original packet. The last signal lbs_eop is the mark of the last cell. We can use these signals to reassemble the cells and specific methods are mentioned in Algorithm 2.
4) *Self-routing header*: It is just the output group number OG shown in Fig. 4 and it will be dropped after passing through Router. In fact, it is the recoding of the output header. Analogously, lbs_active proves the cell active and lbs_dst gives the destination. Reviewing the 2×2 basic sorting unit introduced in the beginning, if the state is conflict, signal lbs_priority will make the decision.
5) *Load balancing header*: Being similar to self-routing header, lbs_active_mid is the significance bit and lbs_priority_mid indicates the priority. The only difference between the two headers is that the destination shown by lbs_dst_mid is the middle group number MG in Fig. 4. Load balancing header is set according to Algorithm 1 and it will be discarded after passing through Balancer.
6) *Beginning mark header*: Being simple but necessary, it is used to show the very starting of the frame after assembling.

C. *Functional Simulation Platform for Sub-modules in UDP*

The design and functional realization of a module need to pass the test of simulation software firstly. Our simulation platform is built with the help of three main tools: the Perl scripting language, Makefile scripting language and the simulation software Modelsim. Perl is used to generate kinds of standard Ethernet frames. Makefile can greatly simplify the work of building a simulation platform and Modelsim makes it possible for us to study the design details visually.

Now, we will introduce the design and verification in detail by taking the key sub-module Assemblage as an example.

Assemblage stands in sharp contrast to what's happening in Splitter and Fig. 6 illustrates its complicated design structure. Three state machines FSM1, FSM2 and FSM3 are responsible for cell processing with plenty of FIFOs to store temporary data.

For the example of OG2, the assembling of cells is shown as follows. On the left of Fig. 6, cells arrive from 8 lines simultaneously, which turn into serial cells after passing through Cell_buffer. But these cells may come from different source ports and are very likely to be scrambled after switching. The state machine FSM1 is designed for cell identification and cell classification. By means of assembling header, we can readily pick out which input line the cell belongs to and then classify the cells into Cell_fifo0 to Cell_fifo15. The most difficult part remains to FSM2 to complete it. We use the similar four state machines to assemble every four groups of classified cells into original data. Note that, here we have no more need for the packet header and the padding bytes. In the last step, FSM3 outputs the packets with the attached beginning mark headers from Pkt_fifos, whose situation monitored by Token-bins.

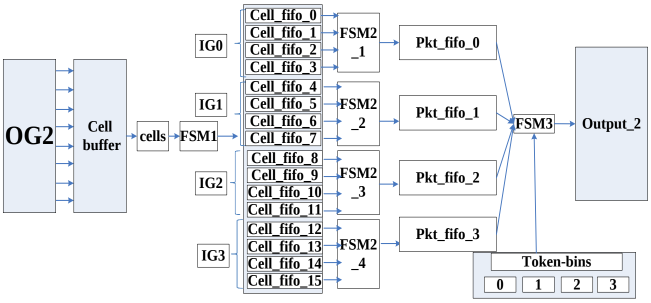


Fig. 6 the design of sub-module Assemblage

As shown in Fig. 7, inputs are serial cells, which can be distinguished by observing signal in_wr and signal in_ctrl. The cell in the circle includes padding bytes, and obviously, it is the last cell of the packet. Based on this information, we can speculate that there is an arriving packet, which consists of seven cells. The output waveform (shown in the rounded rectangle) conforms to the egress rules and proves our judgment.

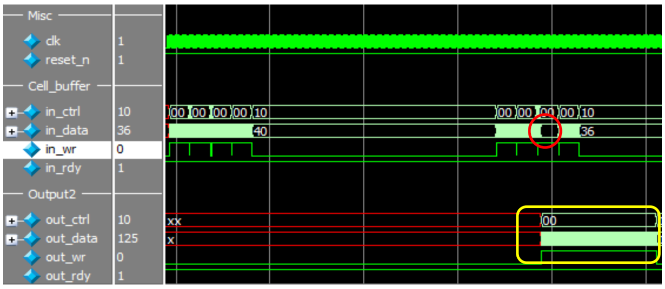


Fig. 7 the simulation waveform of Assemblage

Clearly shown in Fig. 8, the first byte of the output packet is 8`b0000_0000 and corresponding control signal is 2`b11. Undoubtedly, it is the beginning mark header.

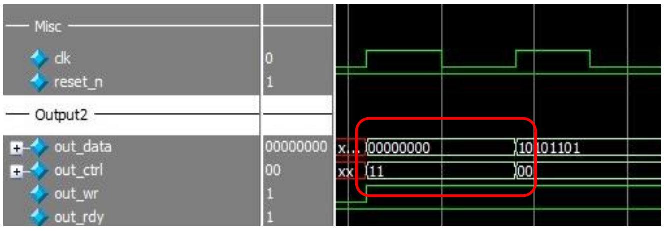


Fig. 8 the starting label header

D. Design and Analysis of Register System

Another major component of the Load-Balanced Multipath Self-routing Switching System is the register system, which provides an interface to the outside world for the UDP system. In practice, we need a mechanism to control the operations in UDP. And when carrying on back-end design and system debugging, we are looking forward to a window to monitor key signals such as the state of FIFOs, the value of counters and so on. Without register system, we can hardly finish the following work for a big project like ours.

1) *the Structure of Register System*

Our register system references the pipeline architecture in the NetFPGA program conducted by Stanford University [11]. As is shown in Fig. 9, every sub-module in UDP connects with a general register, called Generic_reg. All the general registers and another register Regs_master form a ring end to end. And then it can interact with the host computer through the Avalon bus devised by Altera.

The sub-module communicates with the general register through a logical interface (shown by the dark arrow in the picture). When the host wants to access a register, it sends an interrogation signal to Regs_master through the Avalon bus. Then the general registers will pass the signal one by one. In this case, the signal will go through all the registers but only one could respond because of the exclusive destination information in the signal. A register can only accept the request message which belongs to it by checking the destination address. If valid, it replies immediately and transmits the answers backward until back to the host.

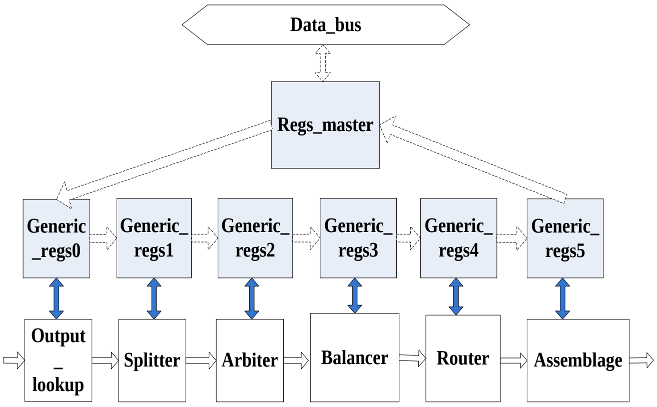
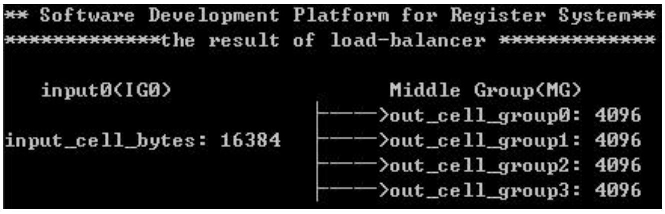


Fig. 9 the whole structure of register system

2) *Software Development Platform for Register System*

Software development platform is based on the tool System Console which is used under the environment of Quartus. On the platform, we mainly perform two tasks with the help of Tcl scripting language. On the one hand, it configures the sub-modules in UDP and the TSE IP core; on the other hand, it extracts the internal signals of the UDP system to help us debug the system. Every sub-module in UDP has its own debugging interface. Taking the Balancer as an example, Fig. 10 shows the pivotal function of our system: load balancing. There are 16384 bytes incoming from IG0. And then, the data is divided into four parts of the same size: 4096 bytes (see the right part of Fig. 10).



```
** Software Development Platform for Register System**
*************the result of load-balancer *************

   input0(IG0)                       Middle Group(MG)
                              |---->out_cell_group0: 4096
input_cell_bytes: 16384       |---->out_cell_group1: 4096
                              |---->out_cell_group2: 4096
                              |---->out_cell_group3: 4096
```

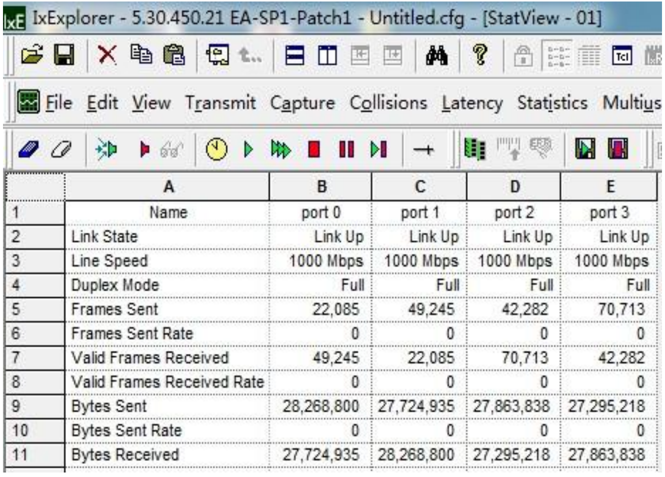
Fig. 10 the result of Balancer

IV. SYSTEM TESTING WITH REAL NETWORK TRAFFIC

Having implemented the Load-Balanced Multipath Self-routing Switching System on FPGA, next we should test it with real network traffic.

IXIA 400T network tester is our leading network test instrument. We use four test modules of all the interfaces on the test board, which can generate or capture standard Ethernet frames transmitted at the rate of 10/100/1000 Mb/s. It is so powerful that we can set, if we want to, every byte of a frame to be sent and get detailed and comprehensive information about the frames captured. The tester also provides remote management capabilities. And coupled with the automated platform set up by Tcl scripting language, we can implement remote automated testing.

Fig. 11 shows us the final statistical result of a test for four ports. According to our configuration, port 0 prepares to receive the output data sent from all the four ports (include itself). Port2, port3, and port4 follow the same way. We can see that there is no data dropped at each port in the case of Line Speed 1000Mbps.



IxExplorer - 5.30.450.21 EA-SP1-Patch1 - Untitled.cfg - [StatView - 01]
File
Edit
View
Transmit
Capture
Collisions
Latency
Statistics
Multius

	A	B	C	D	E
1	Name	port 0	port 1	port 2	port 3
2	Link State	Link Up	Link Up	Link Up	Link Up
3	Line Speed	1000 Mbps	1000 Mbps	1000 Mbps	1000 Mbps
4	Duplex Mode	Full	Full	Full	Full
5	Frames Sent	22,085	49,245	42,282	70,713
6	Frames Sent Rate	0	0	0	0
7	Valid Frames Received	49,245	22,085	70,713	42,282
8	Valid Frames Received Rate	0	0	0	0
9	Bytes Sent	28,268,800	27,724,935	27,863,838	27,295,218
10	Bytes Sent Rate	0	0	0	0
11	Bytes Received	27,724,935	28,268,800	27,295,218	27,863,838

Fig. 11 the statistic views of IXIA

V. CONCLUSION

This paper proposes a new multipath self-routing fabric by merging the Multistage Interconnection Network (MIN) and concentrators. Using the same two MSSS, we construct the Load-Balance Multipath Self-routing Switching Structure and implement the system model on an Altera StratixIV FPGA. After testing under kinds of network environment, we preliminary confirmed that our system can support QoS applications for Next Generation Network (NGN).

During the process of system implementation, we first devised the overall system structure and then the two main constituent parts: the UDP system and the register system. When introducing the UDP system, we analysed the functions of every sub-module and give the expatiation of Assemblage. When presenting the register system, we mainly explained the data path and the software platform.

So far, our system is based on the MSSS (M=4, G=8). And next step, we plan to increase it to M=8, G=16. Meanwhile, the design of large-scale wire-speed multicast base on Load-Balanced MSSS we constructed will be the focus, which needs more excellent design and more thorough support system.

ACKNOWLEDGMENT

Qian Zhan thanks the valuable feedback from Le Yang (Depaul University, Chicago, USA) during the preparation of this paper.

REFERENCES

[1] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, Farnam Jahanian, Internet Inter-Domain Traffic, ACM SIGCOMM 2010;

[2] John Evans, “QoS Decomposed: The Components of the QoS Toolkit”, BRKIPM-2010, Cisco Networkers 2007 Conference;

[3] C. S. Chang, D. S. Lee and Y. S. Jou, “Load Balanced Birkhoff-von Neumann Switches, Part I: One-stage Buffering,” Computer Communications, vol.25 pp.611-622, 2002;

[4] Y. R. Tsai and C. W .Lo, “Banyan-based Architecture for Quasi-circuit Switching”, IEEE ICNS 2006, pp. 23-28;

[5] He W, Li H, Wang B, et al. A Load-Balanced Multipath Self-routing Switching Structure by Concentrators[C]. IEEE ICC 2008;

[6] S. Nojima, et al. "Integrated services packet network using bus matrix switch," IEEEJ. ofSelectAreasCommun.vol. 5, Oct. 1987, pp 1284-1292.;

[7] Li S Y R. Unified algebraic theory of sorting, routing, multicasting, and concentration networks [J]. Communications, IEEE Transactions on. 2010, 58(1): 247-256;

[8] Li S Y R. Algebraic switching theory and broadband applications. Academic Press, 2001;

[9] Hui Li, Wei He, Xi CHEN, Peng Yi, Binqiang Wang, “Multi-path Self-routing Switching Structure by Interconnection of Multistage Sorting Concentrators”, IEEE CHINACOM2007, Aug.2007, Shanghai;

[10] S. Y. R. Li. Algebraic Switching Theory and Broadband Applications. Academic Press, 2001;

[11] Register system - NETFPGA Developers Guide, Available: https://github.com/NetFPGA/netfpga/wiki/DevelopersGuide