SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 6, No. 3, December 2009, 399 - 407

UDK: 004.738.5.057.4

FPGA Implementation of IP Packet Segmentation and Reassembly in Internet Router*

Marko Carević[1,a], Zoran Čiča[1,b]

Abstract: Advanced packet scheduling algorithms in Internet routers work with fixed size cells. As IP packet length is not fixed, it is neccessary to implement segmentation function for dividing IP packet in fixed sized cells. Also, it is neccessary to reassemble original IP packet on output port of the router. Hardware implementation of these two functions will be presented in this paper.

Keywords: IP packet, Segmentation, Reassembly, Crossbar.

1 Introduction

Internet routers represent a basic component of the Internet network. Capacity of the Internet routers is constantly increasing in order to satisfy growing demands for a higher bandwidth that is required for the customers and the Internet applications. Because of that, routers are becoming more and more complex devices that have to implement support for quality of service, traffic security, multicast, and also simultaneously process a great number of IP packets per time unit. In order to increase the capacity of the Internet router, powerful switches are installed in routers, as well as the advanced algorithms for packet scheduling [1].

Crossbar is the central component of the switch in routers of high capacity. Crossbar enables high speed packet switching between different ports. Packet scheduling controls the crossbar operation. Packet scheduler should enable maximally efficient usage of crossbar capacity. Also, nowadays, scheduler needs to be able to support multicast traffic [2] and quality of service. There are several algorithms for traffic scheduling that achieve high efficiency in crossbar capacity usage such as SGS [3], SLIP [4] etc.

Most of the scheduling algorithms are based on the fixed length cells. Fixed length cells enable efficient usage of crossbar's capacity and efficient implementation of scheduling algorithm, which are important properties of

[1]Department of Telecommunications, School of Electric Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11120 Belgrade, Serbia
E-mails: [a]carevicm@gmail.com; [b]cicasyl@ etf.rs
*Award for the best paper presented in Section *Computers*, at Conference ETRAN 2009, June 15-19, Vrnjačka Banja, Serbia.

algorithms scheduling. But, IP packets don't have a fixed length. Because of that it's necessary, after checking the IP packet validity and finding a class of service, to execute segmentation of the IP packet into the fixed size cells. Segmentation of IP packets enables efficient work of crossbar and scheduling algorithm that controls work of crossbar. Also, on the output port of the router, it is necessary to gather all cells of one IP packet and to reconstruct original IP packet, which needs to be sent further into the network. Goal of this paper is to propose a hardware implementation of IP packet segmentation and reassembly that should be easily integrated in IP routers.

In the second chapter of this paper we will describe the FPGA realization of IP packet segmentation and reassembly, and explain the placement of these functions in the Internet router. In the third chapter, design testing and performance analysis will be presented. We'll give the conclusion in the final chapter.

2 Segmentation and Reassembly

This chapter will describe the implementation of IP packet segmentation and reassembly. First, the position of the segmentation function on input port of the router will be explained, and then we'll give the explanation of the implementation of the segmentation function, itself. In the second part of this chapter, we'll explain the position of the reassembly function on output port of the router, and then description of the reassembly function, itself, will be given. Also we will point out a few differences between segmentation and reassembly functions.

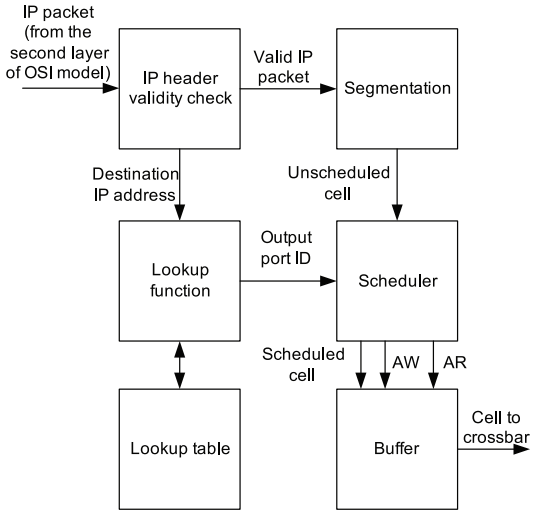

Fig.1 – *Block scheme of the third layer of* OSI *model for input port.*

Fig. 1 shows a simplified block diagram of the third layer of OSI model for input port of the router. As it is shown in Fig. 1, after processing of packets on second layer of OSI model, IP packets appear on the third layer. First, the checking validity of IP header is performed, where inspection for errors inside the IP header is made. During that process, all necessary information are extracted from IP header. These information are necessary for further IP packet processing, such as destination IP packet address, which is necessary for lookup function. All further processing of IP packet is done only if the validity of IP header is verified, else IP packet is rejected as corrupted.

From the IP header check block, IP packet is being sent toward the block that performs the segmentation of packets into fixed length cells. Segmentation block needs to provide fixed length cells for efficient work of scheduler. Scheduler creates schedule for sending cells through crossbar. Because all cells have the same, fixed length, scheduler divides time into slots. One slot represents the time needed for the scheduled cell to be transferred from one router port to another. In order for scheduler to execute its function, it needs a result of lookup function. Based on the result of the lookup function, scheduler knows the output port where the cells should be sent. Also, if quality of the service is implemented, scheduler needs to know which level of quality the packet requires. Field type of service from IP header could be used for determining the quality of service level. After determining the time slot for sending the cell, the scheduler puts cell into a buffer, using the write address *AW*. Using the read address (*AR*), scheduler reads the cell that should be sent to crossbar, from the buffer in the time slot for sending that cell.

Segmentation function, besides dividing packet into fixed size cells, needs to provide additional space inside cells for header in order to make cell processing more efficient. Header has to contain information about demanded quality of service for scheduler's function (packets which demand lower delays need to have a higher priority compared to those that don't have such a demands). Further more, identification of source port for specific cell needs to be stored in the header. This information is necessary on the output port in order for correct packet reassembly, as the cells from different packets can come in consecutive time slots as the result of scheduling algorithm. In this way, based on the source port information in the header, reassembly function can differentiate cells that belong to different packets (that are sent from different input ports). Other solution would be for the scheduler to send that information to output ports, but in case of routers with a large number of ports, this could have negative impact on the efficient router design. Also, the last cell indicator bit is put in the header of every ell. This indication bit makes very easy detection of last cell on output port, so that the complete packet reassembling process could be started. Additionally, since there is information about quality

of service level in the header, it is impossible to make an error to reassemble IP packet from cells that originate from the same input port, but have different quality of service level, which means that cells belong to different packets.

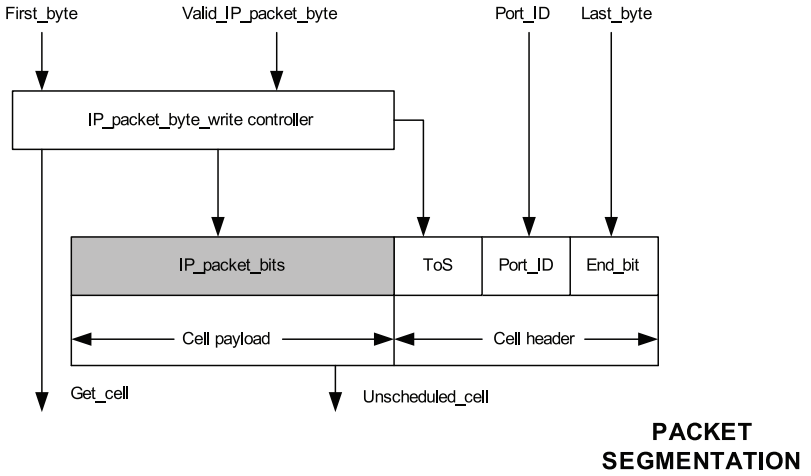

Fig. 2 – *Block scheme of* IP *packet segmentation block.*

Implementation of segmentation function is done in vhdl language. Logical block scheme of segmentation function is given in Fig. 2. Block itself is very simple and is formed of two parts – register in which the cell is placed and controller that controls the writing of the content into the cell. Register itself represents structure of the cell which consists of payload part for the IP packet bytes and header part where the cell's header is placed. Cell's header is 16 bits long and it consists of bit *End* which signalize that it's a last cell of the packet, *Port ID* which represent ID of the input port (port where the cell is created), also field *ToS* which represents type, i.e. class of service. It is assumed that *Port ID* is 10 bits long, and *ToS* 5 bits long. These values enable usage in routers that have up to 1024 ports and up to 32 different service classes. Controller for writing bytes puts IP packet bytes into cell until payload part is full and then sends cell to crossbar scheduler and signalizes to scheduler to get cell using *Get_cell* signal. During the creation of the first cell, from IP header, the value of type of service field is remembered and that value is put into the corresponding place in the cell header (*ToS* field) of each cell of the packet. Beginning of the IP packet first cell's forming is started via signal *First_byte. Port_ID* is defined with hardware position of input port in router, and that value is constant. When the last byte of IP packet is signalized i.e. the end of IP packet (via signal *Last_byte*) bit *End* is set to value 1, in order to signalize it's the last cell of segmented IP packet.

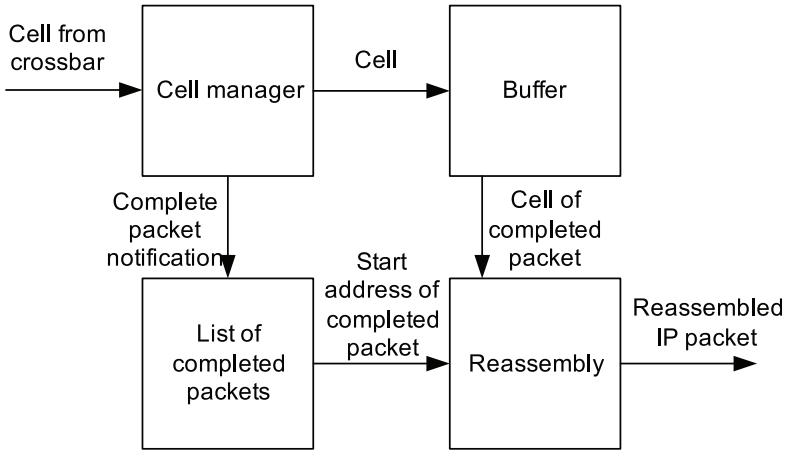

Fig. 3 – *Block scheme of the third layer of* OSI *model for output port.*

Fig. 3 gives a simplified block sheme of the third layer of OSI model for output port. On the third layer of the output router port only the reassembly of IP packets should be done, and then reassembled packets should be passed to the second layer of OSI model. It's obvious that input port is more complex by its functions than the output port. But, output port is dealing with a problem of cell arrival from diferrent input router ports. Hence, output port has to take care of combining adequate cells. This job is enabled by cell's header in which, based on identifiction of input port from which the cell has arrived and level of service quality, output port can determine to which IP packet does the cell belong to. Also, based on the last cell indication bit, output port can determine when some packet is completed. Cells alone are kept in the buffer, and it is important to maintain the information about location of the cells in the buffer, as well as the information of the completed packets that need to be reassembled. Reassembly of packets themselves is a simple function and it consists of reading cells of completed packet from buffer and removing cell's header. IP packet's bytes are then forwarded to the output port second layer of OSI model. Last cell of the packet could be padded with zeros if the packet bits couldn't fill the complete payload part in the cell. But since the reassembly block extracts the information about the packet total length from the first cell of the packet that contains the IP header, padding zeros won't be passed to the second layer of OSI model.

Logical block scheme of reassembly block is given in Fig. 4. Block itself is realized in vhdl language. There are three main parts of reassembly block: selector, address generator and block for removing cell's header.

Selector is used for choosing a completed packet in case there are several classes of service i.e. more than one level of priority. Fig. 4 shows a case when there are two classes of service, higher and lower priority. Signals *Empty_list_CPHP* and *Empty_list_CPLP* give the indication about the existence of completed packets of higher and lower priority, respectively. Selector chooses packet from the list which is not empty and in the case when both lists have packets, then packet is chosen from the list with higher priority. By selecting appropriate list, request for reading from selected list is sent by activating signal *Read_list_CPLP* or *Read_list_CPHP*. From memory, where the selected list is placed, information about starting address of completed packet which will be reassembled is retrieved. The lists themselves are placed in FIFO memories because the completed packets with same priority are served by FIFO principle.

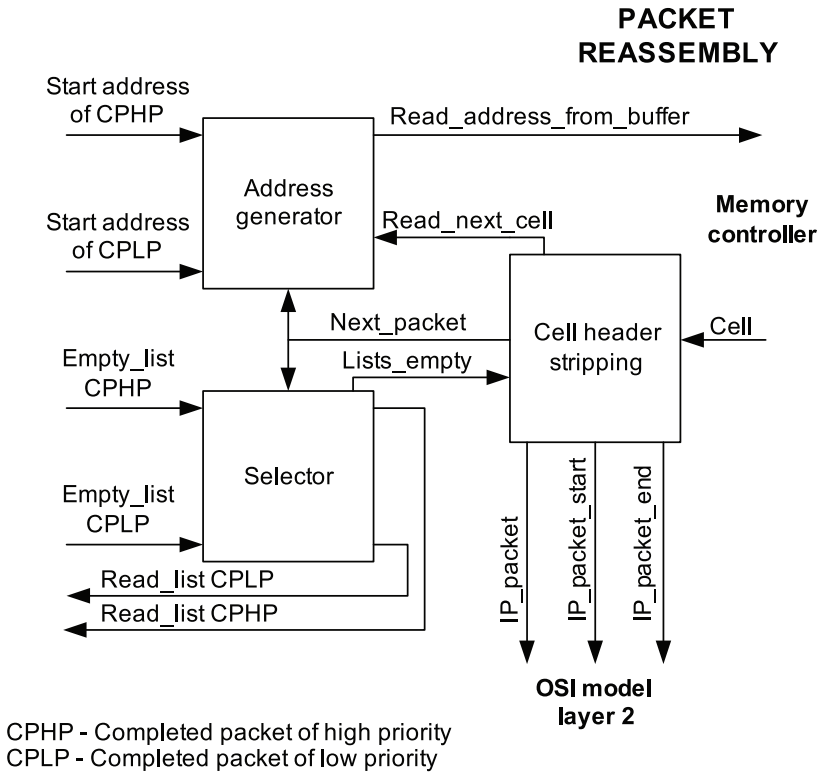

Fig.4 – *Block scheme of reassembly block.*

Block address generator receives the starting address of the packet Address generator is responsible for reading data from buffer where the cells are placed. Buffer itself is placed in external memory (in this work SDRAM memory is

used). In order to optimize and speed up implementation, memory where the cells are kept is divided into blocks. Positions of these blocks are defined based on the packet priority and input port. These blocks are defined as circular buffers, and in this way the result achieved is that cells of the same packet are located at successive locations, which simplifies the work of address generator. This also speeds up the design. After reading the first cell of one packet that is in process of reassembling, address generator receives command (*Read_next_cell*) from the block that removes cell's header for all the other cells of that packet.

Block for removing cell's header also does the reassembly of packets. This block forwards bytes of the IP packet to the second layer of OSI model, without cell's header. By using a signal *Read_next_cell* it dictates the tempo for reading cells from the buffer so there can't be an "empty walk", pause in sending the IP packet bytes to the second layer of OSI model. Sending first byte of the packet is signalized with *IP_packet _start*. At that time, using information from the first cell's header about length of IP packet, the counter that counts the bytes that have been sent to the second layer of OSI model is set. Using this counter this block knows when he has sent the last byte of IP packet which is signalized to the second layer of OSI model with signal *IP_packet_end.* When one IP packet is completely sent, with signal *Next_packet* blocks selector and address generator are signaled that they can start reading the next completed IP packet in buffer, and to send first cell to the block for removing cell's header. In case there are no completed packets (signal *Lists_empty*) block for removing cell's header won't activate signal *Next_packet,* until signal *Lists_empty* is deactivated i.e. until some IP packet is completed.

3 Simulation Results

Design of segmentation and reassembly of IP packet is done in vhdl language. Implementation and design testing is done in Altera's Quartus 8.1 Web Edition environment [5]. Also, hardware verification of design is performed on Altera's developing board DE2, which contains FPGA chip Cyclone II EP2C35F672C6 which is low-cost FPGA chip. SDRAM 8MB memory is used as the buffer during hardware testing. Segmentation and reassembly functions were tested separately. In this chapter we will give the short description of testing on hardware. Also performance analysis for several cell lengths (L) is given at hte end of this chapter.

For the testing of the segmentation function, simple IP packets generator was created in FPGA chip. It was realized using internal ROM memory of the FPGA chip, and its content was filled with the values of several valid IP packets which were sent toward the segmentation block. Cells themselves are placed in the SDRAM memory on adequate locations. By checking content of SDRAM

locations, and also by watching signals on logic analyzer, correct work of segmentation block was verified.

Table 1
Implementation results of segmentation function.

Cell's Length	No of logical elements	No of registers	f_{max}[Mhz]
256	588	505	218.82
512	1215	1018	193.16
1024	3404	2043	162.28

Table 2
Implementation results of reassembly function.

Cell's Length	No of logical elements	No of registers	f_{max}[Mhz]
256	783	563	189.57
512	1465	1076	174.09
1024	2838	2101	157.06

In a similar manner, reassembly function was tested In this case, in the SDRAM memory several IP packets were written, and reassembly block was supposed to read cells of every packet and to reconstruct original IP packet content. Reconstructed content is compared to expected values which were placed in the internal ROM memory of FPGA chip. In this way we verified correct work of reassembly function.

Segmentation block design, as well as the reassembly block design was tested for 3 values of cell's length (L): 256, 512 and 1024. **Tables 1** and **2** give basic information on used FPGA resources, whereas maximum design speed is given in **Tables 1** and **2** for the segmentation and reassembly block, respectively. Design is implemented on FPGA chip Cyclone II EP2C35F672C6.

As it could be seen from **Tables 1** and **2**, design is better as regards used FPGA resources and speed as the cell is smaller. But, the smaller the cell is, the more cell header bits are transferred per IP packet so the usage of crossbar's bandwidth is decreasing, which causes lower router's performances. It's obvious that a compromise should be made between these two opposed demands where in process of deciding parameters such as communication speed between router ports and crossbar, router capacity, buffer access time, etc should be taken in consideration.

4 Conclusion

This work explains the implementation of segmentation and reassembly of IP packets in the Internet routers. These two functions, although rarely mentioned in literature, are very important for efficient work of the router. As we have seen from implementation results, cell's dimension influence in two different manner on the efficiency of the router – at one side, the smaller the cell is, the design is more efficient regarding resources and speed, but on the other side, the bandwidth usage is lower when the cells are smaller. So it is necessary to find good compromise between these two opposed demands when implementing segmentation and reassembly function in the Internet router.

5 References

[1] M. Petrovic, A. Smiljanic: Optimization of the Scheduler for the Non-blocking High-capacity Router, IEEE Communication Letters, Vol. 11, No. 6, June 2007, pp. 534 – 536.

[2] M. Blagojevic, A. Smiljanic: Design of Multicast Controller for High-capacity Internet Router, Electronic Letters, Vol. 44, No. 3, Jan. 2008, pp. 255 – 256.

[3] M. Petrović, A. Smiljanić: Design of the Scheduler for the High-capacity Non-blocking Packet Switch, IEEE Workshop on High Performance Switching and Routing, Poznan, Poland, June 2006.

[4] N. McKeown: The iSLIP Scheduling Algorithm for Input-queued Switches, IEEE/ACM Transactions on Networking, Vol. 7, No. 2, Apr. 1999, pp. 188 – 201.

[5] www.altera.com