Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp1016-1019)

Design of High-Speed Data Buffer Based on FPGA for Gigabit-Rate Exchange Devices

Wang Jie
Department of Computer Science and Technology in Harbin Institute of Technology
China
Ji Zhen-zhou
Department of Computer Science and Technology in Harbin Institute of Technology
China
Hu Ming-zeng
Department of Computer Science and Technology in Harbin Institute of Technology
China

Abstract: With the increase of Internet bandwidth and the development of Internet applications, gigabit exchange devices are used widely. The reasonable design of high-speed data buffer is a key to break throughput rate necklace. We provide a new design of multi-level buffer structure based on Field Programmable Gates Array (FPGA). Parallel Schedule algorithm increases packets transmission speed. By improving pipeline the structure can be applied for ten-gigabit-rate environments.

Keywords: FPGA; Data Buffer; Multi-level buffer; Parallel Schedule

1 Introduction

The bandwidth of high-performance network interfaces has often exceeded the capabilities of workstations to process network packets[1]. The throughout rate is one of the primary bottlenecks for high-speed network exchange devices.

Content analysis of packets transmitting is emphasized more and more. If more information is needed, packets have to be stored. So a good design of high-speed data buffer is very important for high-speed network devices, such as gigabit routers, gigabit switches and so on, to reach approximate linear transmission with mirror delay.

Private hardware logic can reduce CPU loading greatly, and now is used in many applications especially for huge date flow process. We present a high-performance structure for high-speed data buffer based on FPGA with external RAMs.

2 Buffer Implementation Strategy

According the difference of RAMs used for data buffer, there are three common implementation ways: on-chip buffer, off-chip buffer, and union buffer.

2.1 On-Chip Buffer

With the development of ASIC techniques, more and more high-speed memory units are integrated in a complex chip, such as Block RAMs in FPGA. In some direct-transmit systems, only the header of the packet should be buffered, so small RAMs on-chip is enough.

But to most advanced exchange device,

store-transmit is still necessary for deep level filter. So now RAMs on-chip are still too small to complicated designs. On-chip buffer is used in relatively simple modules.

2.2 Off-Chip Buffer

Off-chip buffer means packets are stored in SRAMs(such as ZBT RAMs) or DRAMs(DDR) connected with the main chip. This method has been widely used in many traditional applications.

Obviously, memory accesses trouble the transmission speed. Memory controller logic can not work full duplex. To high-speed communication, memory is so busy, but data process logic has to wait and wait.

Maybe CAM is a good substitute, but it is still too expensive. And CAM need too many pins and higher power consumption that handicaps the overall performance.

2.3 Union Buffer

To utilize the advantage of the two methods above-mentioned, we adopt union buffer that means the header of a packet stored in RAMS on-chip and the remainder enter RAMs on-chip. Some new applications have implemented this schema, such as the design of EECS150[2] showed in figure 2-1, three ports Ethernet Switch, by Berkeley in 2004. In this design, RAMs on-chip work as nibbles FIFOs.

It should be pointed that, RAMs on-chip can act a more important role not if organized well. Another question is that high-speed ZBT SRAM can replace DDR if external buffer is not too huge. The following chapter will discus in detail.

3 Multi-level Buffer Structure

Generally speaking, pipeline operations are necessary for buffer control. Multi-level buffer make simplify every module, and accommodate vast flow sharp change. Figure 2-2 shows the whole structure.

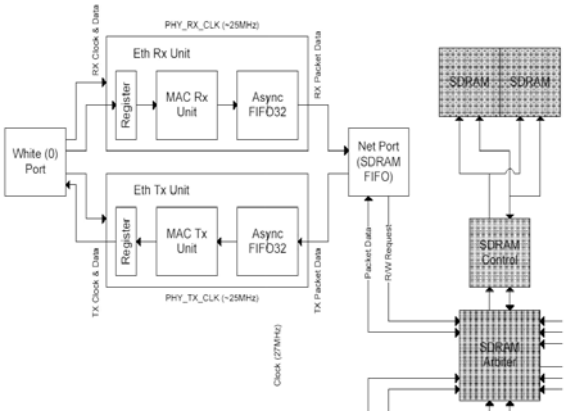

Fig.1 the design of three ports Ethernet Switch Structure (partial)

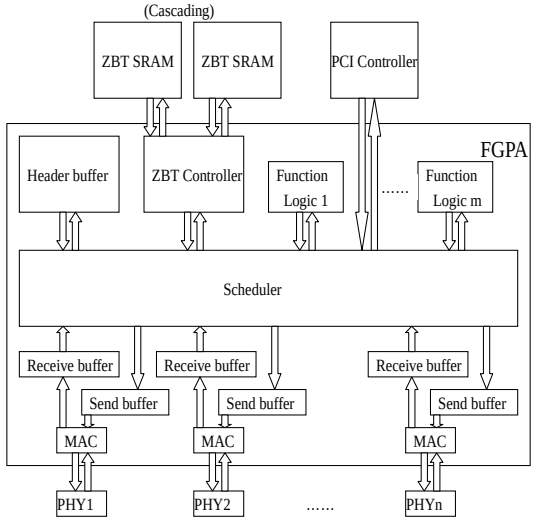

Fig.2 multi-level buffer structure

3.1 Receive Buffer and Send Buffer

Receive buffer and send buffer are constituted with full-duplex Block RAMs. They differ from the nibble FIFOs in MAC, and store the whole packet. They connected corresponding MAC with high-speed WISHBONE BUS[3] (We recommend it).

MAC works slower than Buffers because MAC interface data width is only 8-bit, and WISHBONE bus is usually 32-bit. So when the length of packet mod 4, the remainder is not zero, how to use the free part of a double word to limited on-chip RAMs ?

Throwing is a good way. Suppose M means memory usage ratio.

.Worst case

The length of every packet is 65 Bytes, and its description is 4 Bytes. It needs 18 units. So

$$M_w = (65+4)/(18*4) = 95.83\% \qquad (1)$$

. Average worst case

Only 4 kinds of packet exist with the same probability and their length is 65, 66, 67 and 68 Bytes.

$$M_{aw} = 25\% * ((64+4)/(17*4) + (65 + 4)/(18*4) + (66+4)/(18*4) + (67 + 4)/(18*4)) = 97.92\% \qquad (2)$$

. Actual case

$$M > M_{aw}$$

Thus M is enough high and extra expense for align is not needed.

Receive buffer and send buffer would not be the a bottleneck except they are always full, so scheduler is leading role.

3.2 Header Buffer

If all packets are stored in ZBT SRAM, scheduler only process one receiving packet or one sending packet, because ZBT is sharing resource. Then memory access becomes the bottleneck.

Actually, deep level filter only cares the header of the whole packet. So we can put the header part(For example,128 Bytes) in Header Buffer which is comprised in the same way by Block RAMs. Because Block RAMs can be accessed full duplex, and many small packets need not enter external ZBT SRAM, Scheduler can parallel process packets from different directions.

3.3 Scheduler

Scheduler is the core of multi-level buffer including sending controller and receiving controller. So schedule strategy must be enough flexible and balanced. We give a parallel schedule algorithm with weighted polling for buffers as followed:

(1) If scheduler is free, turn to (2); else turn to (6);

(2) The sending controller tries to lock ZBT control if needed, and receiving controller does so. Turn to (3) and (4);

(3) At the next cycle, sending controller choose an export according to return value of function module (router). Scheduler gets data from Header buffer or ZBT SRAM. (Send buffer is prior to receive buffer.) Turn to (1);

(4) If receiving controller locks ZBT SRAM successfully, it chooses an entry, turn to (5); else turn to (6);

(5) If one receiving buffer is full, it is priority is the highest; and if more than one is full, polling works. If no one is full, the one whose free space is the smallest is prior; and if they almost equal, polling works too. At the same time, data are stored in Header Buffer or ZBT SRAM. Turn to (1);

(6) If the packet is not (or not to be stored) in external ZBT SRAM, another opposite transmission can be started. Turn to (1);

(7) Choose a small packet to process opposite transmission if it can be founded. Else turn to (8);

(8) Waiting until free or (7) can be process. Then turn to (1).

Certainly scheduler also manages other function modules. Because it always chooses a packet from Header Buffer to function modules, and function modules maybe scheduled orderly, pipeline is easy to implement only if time complexity of each module is approximate equal.

3.4 Function Module

Function module can be design as router, state inspect, content filter, and so on. They are transparent to buffers. Only scheduler organizes them and provides data. If one module process frequently, its time complexity has great effect in delay of packet transmission and buffer loading. So excessively complex modules should be simplified and segmented.

4 Hardware Implementation

Following the above schema, we design a four ports general gigabit-rate full-duplex process platform base a FGPA chip, Xilinx Vertex2 XC2V3000 BG728 -4. It included three function modules: address route, state inspect and rule match. Estimated frequency by synthesizing is at least 143.5MHz, higher than the required frequency 125MHz at gigabit-rate speed.

Each function nodule processed a packet in at most 40 clock cycles, much lower than the time MAC finished a packet transmission (For 64 Bytes packet, at least 42 clock cycle including WISHBONE BUS communication). Buffers would not get into a jam even if all packets were stored into ZBT SRAM, because it took almost the same time for function modules to processed different length packets.

To exchange devices, space of send buffer should equal to space of receive buffer. But in this way, it will be coordinating to deal with possible jam if the previous is bigger.

Tests showed when four gigabit-rate ports were full loading with 64 Bytes packets, no packets were rejected.

5 Discuss

If one function nodule is designed to filter content, external mass memory will be necessary whatever the algorithm is. So memory access will be frequent, and centralized filter will consume overmuch cycles. To avoid reduce overall performance Filters can be set distribute to MAC, receive buffer or send buffer.

If half duplex is compatible, MAC will be more intricate[4] and speed will go down. But that will not affect other parts much. With the development of network technique, half duplex will fall into disuse sooner or later.

Although buffers are not full generally in full-duplex mode, flexible flow control strategy is still necessary[5].

Now ten-gigabit network becomes a hot spot. The schema can adapt faster environment without much modifications. If Pipeline design improves in advanced chips, frequency will increase enough high (156.25MHz) to cope with ten-gigabit-rate data flow.

6 Conclusion

The paper presents a design of general gigabit-rate data buffer structure in high-speed exchange devices. The schema of multi-level buffer structure is flexible and credible. Parallel schedule algorithm increases speed of packet transmission to the utmost.

Its availability was verified by hardware implementation of four ports process platform. Trough pipeline improvement, the structure can be widely applied in advanced high-speed exchange interfaces even at ten-gigabit-rate speed.

References

[1] Peter Bellows, Jaroslav Flid, Tom Lehman: GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing, *FCCM'02*

[2] EECS150 Spring 2004 Final Project Voice over Ethernet Switch, Greg Gibeling, University of California at Berkeley.

[3] WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Version B3,www.silicore.netnet/wishbone. htm , 2002.9

[4] Igor Mohor: Ethernet IP Core Design Document, www.opencores.org, 2002.10

[5] Yao Jia-kang: Analysis flow control of full-duplex gigabit Ethernet, *Journal of China Civil Aviation University*, Vol. 19 2000.6 40-43.