Memory Management in Output-Buffering Packet-Switch Design

Jun Xu, Reza Sotudeh
School of Electrical, Communication and Electronic Engineering
University of Hertfordshire, UK
x.jun@herts.ac.uk, r.sotudeh@herts.ac.uk

***Abstract*-The most pressing problem in design of a synchronous buffer-memory system in high-speed packet switches is memory bandwidth. If there are multiple packets heading for the same buffer while the buffer cannot consume them simultaneously, some of the packets will have to be dropped. Two approaches are explored to resolve this problem in this paper. One is via improving the buffer-memory architecture, and the other is via replacing clock-based synchronous technology with handshaking-based asynchronous technology. Both approaches are implemented and the results of experiments run to evaluate several aspects of the implementations are compared.**

I. Introduction

Input buffering, centralised (shared) buffering and output buffering are the three best known buffering strategies in packet-switch design. Under input buffering, packets are stored in an independent buffer associated with each input port, at which they arrive. Under centralised buffering, packets are stored in a centralised memory shared by all input ports and output ports. Under output buffering, packets are stored in an independent output buffer dedicated to output port that is their destination.

In a conventional input-buffering packet-switch design, each buffer is implemented as a single FIFO queue and only the packet at the head of such a queue can be transmitted. If the packet at the head of the queue is blocked, all packets behind it have to be blocked wherever their destinations are, which is known as the Head Of Line (HOL) problem. Input buffering packet switches with the HOL problem can only achieve a maximum throughput of around 60% [1].

Output buffering and centralised buffering packet-switches can achieve throughput of around 80%. However, blockage/data loss can still occur when packets from different input ports head for the same buffer. If the buffer cannot consume all the incoming packets at the same time, some of the packets will have to be dropped. The conventional solution to this problem is to increase memory-access-speed or widening data-path. However, it can be observed that in recent years, the bandwidth of links used for interconnection has continued to increase [2]. To further extend the bandwidth of the memory has therefore become increasingly impractical.

In this paper, two approaches are presented to eliminate the bandwidth problem for output buffering packet-switches. One is via adding pipelines prior to each output buffer, and each pipeline is dedicated to one input port. Provided that multiple packets head to the same output buffer simultaneously, the contention is resolved while they are rippling through the pipelines. To avoid data loss, the minimum depth of each pipeline must be N+1, in which N is the number of input ports. Buffer in such a system is constituted by multiple memory banks. Memory addresses are assigned to packets in sequence and on demand. Once memory addresses are assigned, packets from different input ports can be uploaded to their associated memory banks concurrently and independently. Unlike Prizma [3], at which the clock speed in control logic has to be N times faster than the clock speed in data path, the newly proposed system is synchronised by a single clock signal.

Having seen that clock-based buffer-memory systems are struggling to meet modern packet-switch design, a question arises: can a non-clocked system, namely an asynchronous buffer-memory system, avoid the bandwidth problem? The second contribution of this paper is therefore to explore the possibility of implementing an asynchronous buffer-memory system. Under asynchronous operations, synchronisation is devolved to local control signals: handshaking signals, instead of using clocks. As a nature of asynchronous technology, data transfers between two circuits are based on a Point-to-Point flow control protocol, which guarantees that no data loss will occur. The asynchronous approach in this paper shares most of buffer-memory architecture with the synchronous pipeline approach. However, unlike the synchronous approach, contention is resolved by an arbiter [4] rather than in pipelines. The arbiter only allows one request to pass through at a time; the one that arrives first is selected. When two requests arrive simultaneously, it arbitrarily selects one to go through.

The rest of this paper is organized as follows: in Section 2, the architecture of a buffer-memory system that is shared by both approaches is presented; the implementation detail by asynchronous technology is presented in Section 3 and the implementation detail by synchronous technology is presented in Section 4, respectively; simulation results are presented in section 5; finally conclusions are drawn in Section 6.

II. Architecture of Memory Management in Output-Buffering Packet-Switches

Fig.1 illustrates the structure of the proposed buffer-memory system for a 2by2 output-buffering packet switch. The memory in such packet-switches is constituted by multiple memory banks (only two memory banks are illustrated in the figure though more banks could be accommodated if required).

Each memory bank is logically independent from others and is designed to store one packet. Each memory bank has two 2-to-1 multiplexers in front of it: one for data, and the other for control signals.

The control unit interacts with both input ports and assigns memory banks to incoming packets on demand. It builds up links between input ports and memory banks by setting up the associated 2-to-1 multiplexers. Once a link is built up, the control signals and data from the associated input port can directly communicate with the assigned memory bank. A circular counter, implemented in the control unit, records memory bank addresses and determines which pair of 2-to-1 multiplexers should be configured. Although the control unit processes memory-arrangement requests sequentially, once memory banks are assigned to packets, data transfers between the memory banks and their associated input ports can be conducted concurrently and independently.

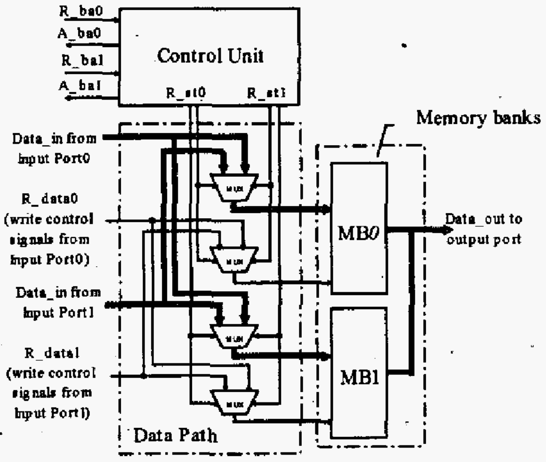


Figure 1 Overview of the Proposed Memory-management System

III. IMPLEMENTATION

A. *Asynchronous implementation*

The asynchronous circuits in this paper are based on a Speed-Independent model, where delays on wires are regarded as zero or negligible while delays on gates are unbounded [5]. Data encoding is based on bundled-data protocol. In the case that data value is n-bit wide, n+2 wires, i.e., n bits for data, 1 bit for request, 1 bit for acknowledgement, are required in transferring each data. Encoding for handshaking signals are based on a 4-phase level signalling protocol (return-to-zero). After each transfer, the channel signalling system returns to the same state as it was in before the next transfer can start.

The asynchronous memory-management is described as the format of STG (State Transition Graph) [6] presented in Fig.2. *R_ba0* and *R_ba1* are the request signals from Input Port0 and Input Port1 respectively. They are asserted to demand the control unit in the buffer-memory to arrange memory before packets can be uploaded. Unlike the synchronous pipeline approach, the asynchronous system resolving contention relies on an arbiter. The arbiter only allows one request to pass through at a time; the one that arrives first is selected. When two requests arrive simultaneously, it arbitrarily selects one to go through. The arbiter is communicated by using the handshaking pair, *R_arbiter0* and *A_arbiter0*, for packets from Input Port0, and *R_arbiter1* and *A_arbiter1*, for packets from Input Port1 respectively. Once granted by the arbiter, setting up the associated 2-to-1 multiplexers is conducted via using handshaking pairs, *R_st0* and *A_st0*, for packets from Input Port0 and *R_st1* and *A_st1* for packets from Input Block1 respectively. The counter, which provides memory bank addresses, is incremented as soon as a memory bank has been assigned. To ensure that packets from different input ports are not uploaded to the same memory bank, the arbiter must not be released until after the counter has been incremented. The counter is driven by the handshaking pair *R_counter* and *A_counter*. *A_ba0* and *A_ba1* are the acknowledge signals corresponding to *R_ba0* and *R_ba1* respectively, notifying the input ports after the associated memory-arrangement jobs have been done.

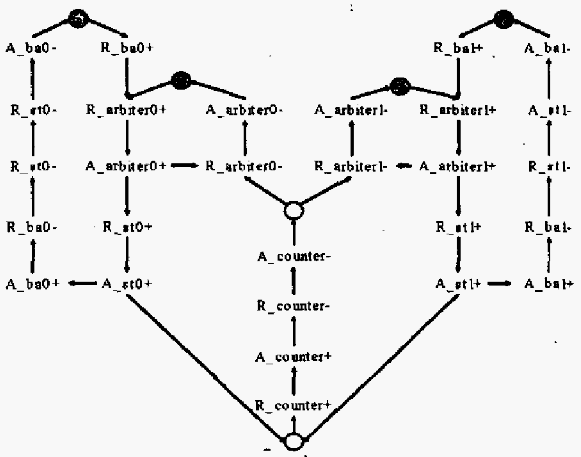


Figure 2 STG for Asynchronous Memory-management

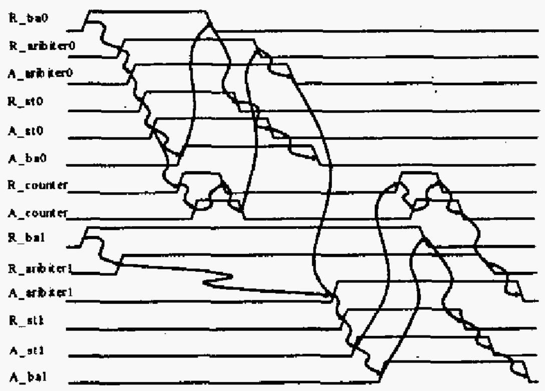
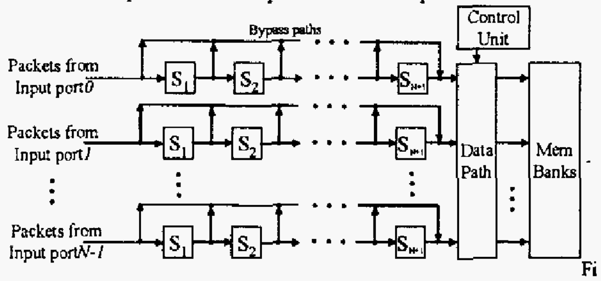


Figure 3 Timing of Asynchronous Memory-management

Further explanation on how the asynchronous system manages its memory when two packets head for the same buffer simultaneously is illustrated in Fig.3. The arbiter randomly grants the request (*R_ba0*) from Input Port0 first, and as a result, a memory-bank is assigned to the packet from Input Port0 prior to the one from Input Port1.

B. Synchronous implementation

In the synchronous approach, the memory bandwidth problem is resolved by using N pipelines before each buffer (N is the number of input ports) as shown in Fig.4 (S_i in the figure represents the *i*th stage of pipeline). Each pipeline is dedicated to one input port. Provided that multiple packets head to the same output buffer simultaneously, the contention is resolved while they are rippling through the pipelines. In order to avoid data loss, the minimum depth of each pipeline must be N+1. A token alternating between "0" to "N-1", implements a fair policy for sequencing requests when there is contention: when the token is "*i*", the packet from Input Port*i* will proceed first, and the packet from Input Port*i+1* will proceed in the second place and the packet from Input Port*i-1* will proceed at last.


Bypass paths
Control Unit
Packets from Input port*0*
Packets from Input port*1*
Packets from Input port*N-1*
S_1
S_2
S_{N+1}
Data Path
Mem Banks

Fi
gure 4 Synchronous Pipelines

To reduce the latency that the pipelines introduced, bypass paths are implemented in each pipeline. Packets only have to ripple through the number of pipeline stages that are enough for the control unit to settle down their memory banks. Once their memory banks are arranged, packets can be directly uploaded from any stage of pipelines. The first packet that wins contention is propagated to its assigned memory bank after rippling through its second stage of pipeline; the second packet is propagated to memory after its third stage of pipeline and the Nth packet will be done after its N+1th stage of pipeline.

For a 2by2 packet-switch, three stages are involved in each pipeline and it takes one clock cycle for packets to ripple through each stage. The token alternates between "0" to "1". Once the token has been used to make such a decision, its value is changed. The counter recording memory-bank address is incremented in the same clock cycle as a link is built up.

An example showing how the synchronous system manages memory is presented in Fig.5. The period from t_{11} to t_{14} illustrates the scenario when packets reach their associated pipelines at different clock cycles. Same as the asynchronous approach, *R_ba0* and *R_ba1* are asserted by Input Port0 and Input Port1 respectively to demand the control unit to arrange memory before packets can be uploaded. The associated links and memory banks are assigned between t_{12} and t_{13} by using *R_st0* and between t_{13} and t_{14} by using *R_st1* respectively. *A_ba0* and *A_ba1* are flagged high after the memory-arrangement for the associated packets has been done.

. The period from t_{35} to t_{38} shows how the system manages memory and resolves contention when two packets from different input ports head for the same output port simultaneously. Both packets are detected by the control unit in the first stage of their pipelines between t_{35} and t_{36} when *R_ba0* and *R_ba1* are both high. Since the token is "0", the control unit builds up the link (by setting up the associated 2-to-1 multiplexers) for the packet from Input Port0 between t_{36} and t_{37}, and built up another link for the packet from Input Port1 between t_{37} and t_{38}. The token is switched to "1" once the contention is resolved at t_{36}. Uploading the packet from Input Port0 into its assigned memory bank starts from the third clock cycle (t_{37}) and the packet from Input Port1 from the fourth clock cycle (t_{38}) respectively.

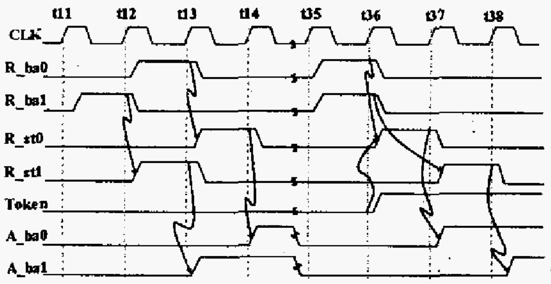


Figure 5 Timing of Synchronous Memory-management.

IV. Experimental Results

A. Simulation environment

A synchronous and an asynchronous 2by2 output buffering switches are implemented for the evaluation of each proposed buffer-memory system. The switches consist of four blocks, i.e., two input blocks and two output blocks. The input blocks identify the destination of packets, and the buffer-memory systems are allocated at output blocks. The asynchronous control circuits and synchronous circuits are synthesized using Petrify[1] and SIS respectively with 0.5μm CMOS technology. The minimum clock cycle for the synchronous circuit based on PSPICE simulation is 6ns. Typically, transmitting one flit through a synchronous switch takes six or seven clock cycles: one clock cycle for receiving the flit at an input block; two or three clock cycles for the flit rippling through its associated pipeline, which depends on if there is contention; and three clock cycles for the flit to be loaded into memory and then forwarded to next switch/destination. A packet in this paper consists of two parts: header and payload. Headers contain routing information while payloads only contain data.

B. Simulation results

The simulation waveform generated by MicroSim Design Centre for the asynchronous approach is presented in Fig.6. Two 8-flits packets from different input ports head to the same buffer-memory simultaneously. As shown in the figure, once memory banks were sequentially assigned, the two packets were concurrently and independently propagated into their

[1] http://www.lsi.upc.es/~jordic/petrify/petrify.html.

associated addresses[2]. *R_data0/A_data0* and *R_data1/A_data1* are the associated handshaking signals for uploading packets into assigned memory banks (refer to Fig.1).

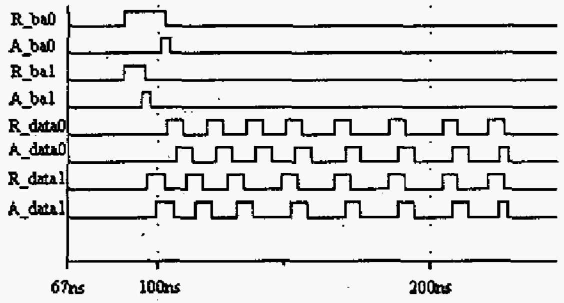


Figure 6 Simulation waveform of propagating two packets into the same buffer-memory

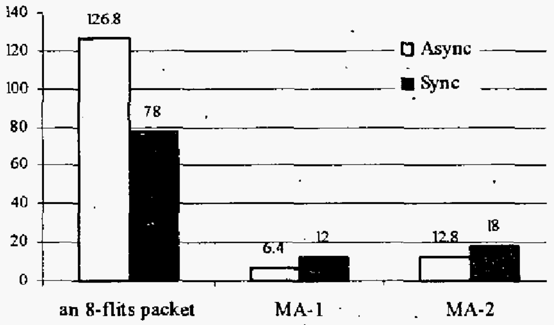


Figure 7 Processing time of the proposed buffer-memory systems and their associated switches

Fig.7 presents another two results. The first result shows that the asynchronous buffer-memory system only consumes half of the time (equivalent to one clock cycle) in managing memory for one packet (MA-1) and two thirds of the time (equivalent to two clock cycles) in managing and resolving contention than the synchronous pipeline approach (MA-2). The second result shows that the synchronous switch outperformed the asynchronous switch and the synchronous switch only spent thirteen clock cycles (six clock cycles for header-transmission, and one clock cycle for each flit of payload) in routing through a packet. The result suggests that although the synchronous pipelines introduce extra latency, the impact on the overall performance of a switch is limited. If flits in a packet are transmitted consecutively in the synchronous approach, the extra latency consumed in the pipelines only applies to the first flit of a packet (header) while the latency of the subsequent flits can be overlapped by their preceding. In the case that packets are transmitted non-consecutively, the latency on pipelines can be reduced by bypassing packets from the pipelines.

[2] It works in the same way as in the synchronous approach, in which once memory-banks are sequentially assigned, packets can be concurrently and independently propagated into their associated addresses.

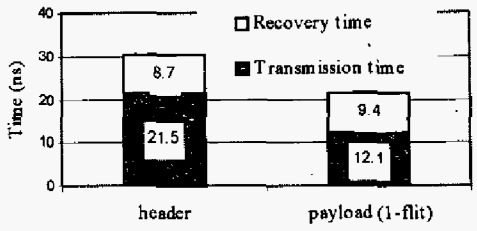


Figure 8 Contribution of the recovery time in the asynchronous transmission cycle for a header and one flit of payload respectively

Fig.8 explains why the asynchronous switch failed to win over the synchronous switch: the asynchronous circuits wasted a lot of time in recovering handshaking signals. As mentioned in the asynchronous implementation section, for a circuit ruled by return-to-zero signalling protocol, after each transfer, the channel signalling system must return to the same state as it was in before the next transfer can start. Fig.8 shows that despite that transmitting a header and one flit of payload through a switch only took about 21ns and 12ns, respectively, the asynchronous approach spent another 9ns each in returning to the same state as it was. The performance of the asynchronous approach can be improved by replacing the 4-phase signalling protocol with a 2-phase signalling protocol. Traditionally, 2-phase signalling protocols do not have the recovery-time problem.

V. SUMMARIES AND CONCLUSIONS

In this paper, two approaches were explored to resolve the memory bandwidth problem for output buffering packet-switches. One is via improving the buffer-memory architecture, and the second approach is via replacing clock signals with handshaking signals. In the former case, contention is resolved while packets are rippling through their associated pipelines. In the latter case, contention is resolved by an arbiter. Both approaches are implemented and compared. The experimental results suggest both buffer-memory management systems can resolve the bandwidth problem. The experimental results also showed that the asynchronous buffer-memory management system outperforms its synchronous counterpart, but the asynchronous switch lost to the synchronous switch due to its recovery time.

REFERENCES

[1] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, Input versus output queuing on a space division packet switch, IEEE Transactions on Communications, COM-35 (12): 1347-1356, December 1987.
[2] C.B. Stunkel, "Challenges in the design of contemporary routers", Proceedings of the 2nd parallel computer Routing and communication workshop, pp 139-152, June 1997.
[3] Minkenberg, C.; Engbersen, T.; "A combined input and output queued packet switched system based on PRIZMA switch on a chip technology" Communications Magazine, IEEE, Vol. 38, Issue: 12, Dec. 2000 PP:70- 77.
[4] C. L. Seitz, System Timing. In C.A. Mead and L.A. Conway, editors, Introduction to VLSI Systems, chapter 7. Addison-Wesley, 1980.
[5] Scott Hauck, Asynchronous Design Methodologies: An Overview, Proceedings of the IEEE, Vol.83, No.1, pp69-93, January 1995.
[6] T.-A. Chu, Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications, PhD Thesis, MIT, June 1987.