Asynchronous Packet-Switching for Networks-on-Chip

Jun Xu[1], Mark B. Josephs[2] and Reza Sotudeh[1]
School of Electronic, Communication and Electrical Engineering, University of Hertfordshire[1]
Faculty of Business, Computing, and Information Management, London South Bank University[2]
x.jun@herts.ac.uk[1], mark.josephs@lsbu.ac.uk[2], r.sotudeh@herts.ac.uk[1]

Abstract

System-on-Chip design is facing increasing challenges in its integration, global wiring delay and power dissipation. Interconnection network technology has the advantage over conventional bus technology in its scalability; on the other hand, asynchronous circuit design technology may offer power saving and tackle the clock-skew problem. Chip designers are thus turning their attention to Network-on-Chip solutions. Packet-switches play a key role in interconnection networks and this paper focuses on their implementation as asynchronous circuits. The results of experiments run to evaluate several aspects of the routing switch implementation are presented.

1. Introduction

As technology scales, a variety of challenges have been presented to IC developers. System integration is one among them. Although buses are still the dominant approach, interconnection networks have been receiving more and more attentions as an alternative integration solution [2, 8, 9, 11]. An interconnection network is comprised of communication links and packet-switches. The links connect hosts to switches and switches to switches. Packet-switches enable data and instructions to be switched from source hosts to any desired destination. One advantage of interconnection networks over buses is the scalability in throughput, latency, cost and integration of the system.

Wire delay is another challenge IC developers have to confront. In the near future, technology scaling will cause wire delay to dominate, while gate delay will no longer be a critical factor in most systems. This dramatic increase in the delay of a global wire, almost doubling every year, affects the signaling, timing, and architecture of digital systems. This makes it extremely difficult to distribute a global clock with low skew [2]. One solution is to devote a large quantity of interconnect metal to building a low-impedance clock grid or wire using new materials (e.g. copper). However, this solution is anticipated to only be effective for one or two more generations [1]. A more radical approach is to eliminate global synchrony. One can either divide the chip into separate clock domains (known as Globally Asynchronous Locally Synchronous), or more aggressively, fully employ asynchronous circuit design technology, such as [4, 7, 14].

Power dissipation has become another critical metric in VLSI circuit design. The growing market of mobile, battery-powered electronic systems fuels the demands for ICs with low power dissipation. Unfortunately, power dissipation in real-life ICs does not follow the descending trend in semiconductor technology [3]. Including asynchronous circuits into a complex VLSI design can help reduce power dissipation [10]: unlike a synchronous system, in which all switching gates charge and discharge with the transition of the clock signal even if they are not in use, consuming power in asynchronous circuits takes place only when a circuit is in operation.

Having seen the challenges in SoC design and the advantages of interconnection-network technology and asynchronous circuit design technology, one question may arise: could the combination of these two technologies provide a solution to the interconnection of SoC. This paper aims to address this question and to consider the feasibility of interconnection network components using asynchronous methods. In particular, the asynchronous design of a packet-switch is examined. The rest of paper is organized as follows: in Section 2, the architecture of a packet-switch is presented; the implementation of the packet-switch by asynchronous technology is presented in Section 3; simulation results are presented in section4; limitations of this work are stated in Section5; finally conclusions are drawn in Section 6.

Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD'06)
0-7695-2556-3/06 $20.00 © 2006 IEEE

2. Pipelined data processing in output-buffering packet-switches

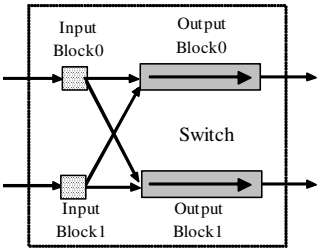

Figure 1. Overview of a 2by2 output buffering packet-switch

An output-buffering packet-switch [13] is proposed for this study as illustrated in Fig.1. To ease our implementation, the switch is only equipped with two input/output ports. The switch consists of four blocks: Output Block1, Output Block0, Input Block1 and Input Block0. Both input and output blocks are further divided into two parts: control block and data path. The input data path can buffer one flit at a time and one flit is defined as 32-bit wide in this paper. The output data path as buffer memory is the main storage element of the switch.

Data transfers between two circuits are point-to-point, involving requests, which initialize each transfer, and acknowledgements, which signal the completion. Before a sender can start its data transfer, the receiver must have indicated the sender that it is ready to take in the data.

Each packet consists of two parts: header and payload. Header is one flit long, located at the beginning of each packet and containing all output ports a packet will pass through. The rest of a packet is payload, only containing data. Packet-size in this paper is fixed.

Packets at each switch are processed in a pipelined fashion as shown in Fig2. Incoming flits from their previous hosts are buffered at input blocks as they arrive. If a flit is the header of a packet, its routing destination is identified, and then a request is sent to the corresponding output block to arrange memory space for the packet. If the flit belongs to payload, it is then forwarded to the same memory as its header. In this switch model, input blocks are involved in stages 1(a) and 1(b), and output blocks are involved in stages 1(c), 2 and 3.

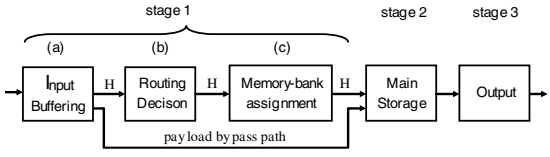

Figure 2. Pipelined processing model

2.1 Input data path

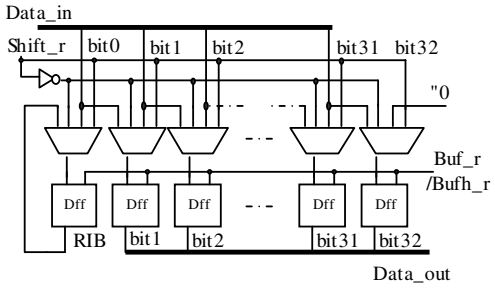

Figure 3. Input data path

As a header arrives at an input block, its routing destination is immediately identified. In a 2by2 packet-switch, routing decisions for a packet can be made using just one bit of information. Here, we assume that the routing information bit (RIB) is always the Least Significant Bit (LSB) at each header. This determines the output block with which to communicate.

Making routing decisions is achieved by means of a combinational operation: shifting and buffering, at the input data path. As shown in Fig.3, the input data path includes 33-bit D-type flipflops (DFF's). If the incoming flit is payload, it is buffered from bit1 to bit32 and then outputted from bit1 to bit32. However, if the flit is a header, it is shifted and buffered from bit0 to bit31 and then outputted from bit1 to bit32. In this scenario, bit32 is filled with "0", turning into the new Most Significant Bit in the header, and bit1 turns into the new Least Significant Bit. These two bits together with the rest of header are then transmitted to their next host. The 33rd bit, bit0, is therefore implemented to accommodate the routing information bit and remove the used routing information from each header.

2.2 Memory Allocation

Memory arrangement for each incoming packet involves operations from both input blocks and output blocks; however, it is mainly conducted at output blocks as illustrated in Fig.4. 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.

2-to-1 multiplexers at each output block are associated with memory banks. Each memory bank has two 2-to-1 multiplexers in front of it, one for data, and the other for control signals. Since each output block communicates with two input blocks, through the multiplexers, signals from either of the input blocks can be propagated to any memory bank. The output control block forms connections between input blocks and memory banks by setting up the associated 2-to-1 multiplexers. A counter,

implemented in the output control block, provides memory bank addresses and determines which pair of 2-to-l multiplexers is supplying data.

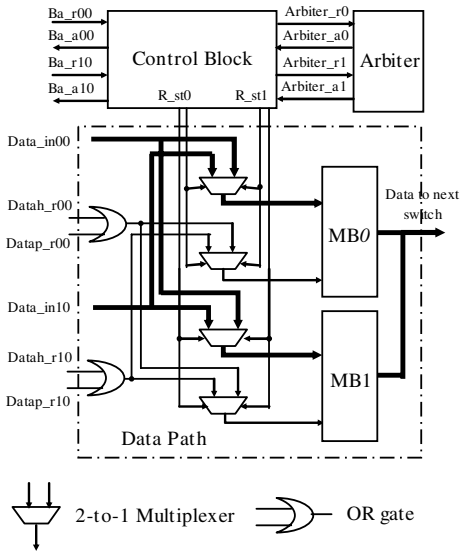

Figure 4. Memory arrangement in Output Block0

To avoid collision, setting up 2-to-l multiplexers for different packets must be mutually exclusive: only one action is allowed to progress at a time, therefore, an arbiter [16] must be employed. The arbiter 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. A request from an input block for memory therefore will not proceed until it is granted by the arbiter. To ensure that no two packets will crash at the same memory bank, the arbiter must not be released until the counter has been incremented to point to the next memory bank.

This memory-management design allows packets from different input blocks to be loaded into the memory concurrently once their addresses have been assigned, thus improving the performance of packet-transmission.

3. Asynchronous implementation

3.1 Asynchronous design methodologies

The asynchronous circuits in this paper are implemented based on a Speed-Independent model. In such a circuit, delays on wires are regarded as zero or negligible while delays on gates are unbounded [12].

The handshaking signals (request and acknowledgement) are encoded as a 4-phase level signaling protocol (return-to-zero). The 4-phase signaling protocol uses the level of the handshaking signals to indicate events. There are four transitions involved in the completion of each transfer. After each transfer, the channel signaling system returns to the same state as it was in before the next transfer can start.

Data encoding is in compliance with a bundled-data protocol (also known as single-rail protocol), which employs one wire for each bit of information. This encoding is used the same convention as in synchronous circuit design. 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 datum from the sender to the receiver.

3.2 Input control logic

Processing headers and payloads at input blocks are described into two separate Signal Transition Graphs (STG's) [5], as shown in Fig.5 and Fig.6 respectively.

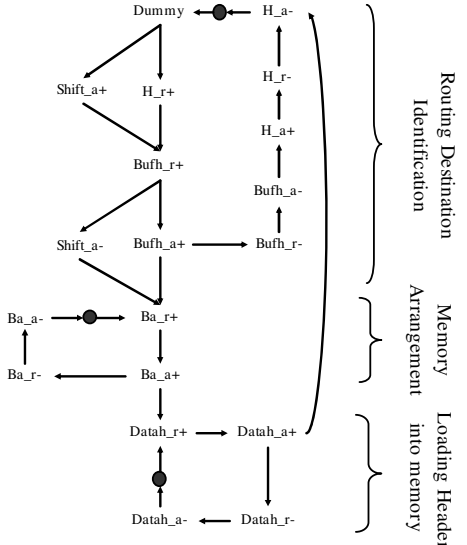

Figure 5. STG for head-processing

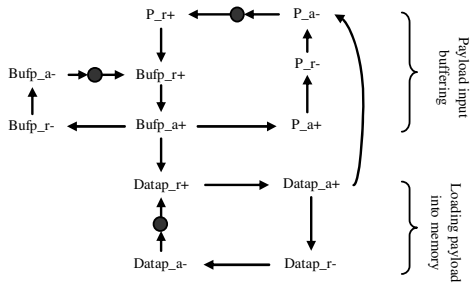

Figure 6. STG for payload-processing at input blocks

H_r and *H_a* are the handshaking pair interacting with the sender for header-transmission. The receiver notifies

the sender whether it is ready to accept a new header using *H_a,* and *H_r* is activated when a new header is asserted.

Shift_r is the shifting request signal, activated only when an incoming flit is the header, and is released once the header is buffered at the input data path. The shifted header is buffered as *Bufh_r* goes high when the routing information bit is sampled. *Shift_a, Bufh_a* are the acknowledge signals corresponding to *Shift_R* and *Bufh_R* respectively.

In order to accommodate the right routing destination information from each packet, the shifting operation must take place before the buffering operation. Since *Shift_a* goes high only after the incoming header has been shifted, the circuitry is safe when *Bufh_r* is triggered only after *Shift_a* and *H_r* both are asserted. The falling transition of *Shift_a* indicates that the routing information has been stabilized in the input block.

Processing payload in an input block is conducted in two steps, i.e., buffering it and then forwarding it to the same output block as its header. The input control block interacts with the sender using the handshaking pair *P_r/P _a. P_r* goes high as one flit of payload is sent. The flit is buffered at the input data path as *Bufp_r* goes high once a high *P_r* is detected. The input block acknowledges the sender its acceptance by driving *P_a* high after *Bufp_a* goes high.

3.3 Output control logic

Memory arrangement in the output control block is described in Fig.7. *Ba_r00* and *Ba_r10* are the request signals asserted by Input Block0 and Input Block1, and *Ba_a00* and *Ba_a10* are the corresponding acknowledge signals, respectively.

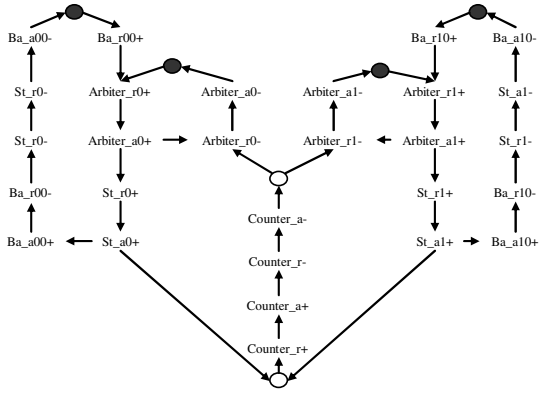

Figure 7. STG for memory-arrangement in Output Block0

The output control block forms the connections between Input Block0 and memory banks by setting up the associated 2-to-l multiplexers using handshaking pairs, *St_r0* and *St_a0,* and for Input Block1 using *St_r1* and *St_a1*, respectively. The counter, which provides memory bank addresses and determines which pair of 2-to-l multiplexers are supplying data, is driven by the handshaking pair *Counter_r* and *Counter_a*. The output control block communicates with the arbiter using the handshaking pair, *Arbiter_r0* and *Arbiter_a0,* for packets from Input Block0, and *Arbiter_r1* and *Arbiter_a1,* for packets from Input Block1 respectively.

The output control block is also responsible for forwarding the packets stored in the memory to their next switches or destination hosts. The operation is described in Fig.8. Packets are read out of memory flit by flit using the handshaking pair, *Datao_r* and *Datao_a,* and then are forwarded to their next switches or destination hosts using the handshaking signals, *Inout_r* and *Inout_a. lnout_r* and *Inout_a* are the handshaking pair employed between switches or between a host and a switch. Signals on *Inout_r* are passed onto *H_r* (refer to section 3.2) when the incoming flit is a header, and are passed onto *P_r* when it is payload. Correspondingly, Signals on *H_a* and *P_a* are multiplexed onto *Inout_a*.

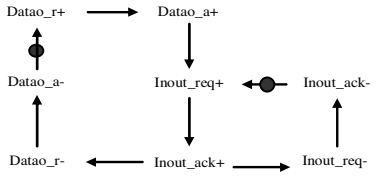

Figure 8. STG for packet-output

4. Experimental results

4.1 Simulation environment

To evaluate the asynchronous implementation, a synchronous packet-switch was also implemented based on the same architecture presented in Section 2. Asynchronous control circuits in this paper were synthesized using Petrify with 0.5μm CMOS technology, and synchronous control circuits were synthesized using SIS. The implementations were evaluated in MicroSim Design Centre. The minimum clock period was determined by the critical path, and a minimum clock period of 6ns was obtained from the PSPICE simulation.

The base system used for the simulation was a k-stage butterfly network [15]. The packet size in the evaluation was fixed to 8-flit, l6-flit, 24-flit or 32-flit long, and the traffic of networks was always unloaded. The interface between a host and a network was viewed as contributing to the same routing delay as a switch [6]. Packets in the interface were stored at a memory, which had the identical performance to the memory in the switch. The

control logic in the network-to-host interface and the host-to-network interface had the same performance as the input control logic and the output control logic in the packet-switch respectively.

4.2 Simulation results

Fig.9 shows that the latency of routing an 8-flit long packet through an unloaded 2-stage network as well as the contributions of header and payload to the overall latency. Fig.10 further shows the performance of each switch in the network in processing each individual flit. The simulation results indicate that although the asynchronous switches outperformed the synchronous switches in processing each individual flit, routing the whole packet in the asynchronous network was slower than in the synchronous network.

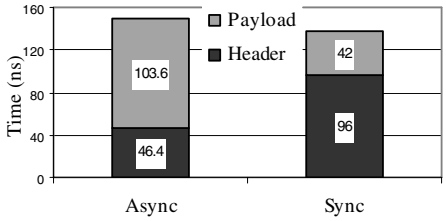

Figure 9. Latency of transmitting an 8-flit packet through a network

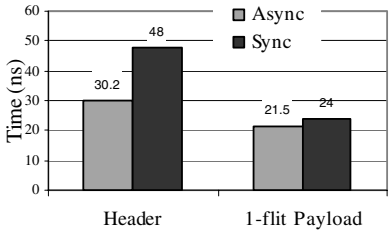

Figure 10. Latency of transmitting 1-flit through a switch

The asynchronous switch won over the synchronous one in transmitting each individual flit was mainly benefited from the way it making progress: the asynchronous circuitry never wasted time in waiting clock transitions like its synchronous counterpart, and always immediately progressed once its communication partner responded.

When flits were transmitted consecutively in a pipeline style in an unloaded network, it was found that the routing time of each flit was overlapped by its neighboring flits. The more they could overlap with each other, the less routing latency they would have. However, it was also found that the impact of the recovery time for the asynchronous circuitry, ruled by a 4-phase level signaling protocol, also became significant. Our simulation result shows that the recovery operation caused the asynchronous pipeline loosely overlapped: only 31% of processing time on each flit was overlapped by its neighboring flits, compared to the synchronous network, in which the payload only contributed seven clock cycles to its overall routing latency.

The impact of packet-size on the network performance is illustrated in Fig.11, in which packets varied from 8 flits to 32 flits routing in an unloaded 2-stage network. The simulation result shows that increasing packet-size weakened the performance of the asynchronous implementation more than its synchronous counterpart. This again can be explained by the sluggishness of the asynchronous pipelining as the result of the recovery operation.

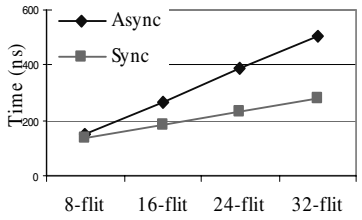

Figure 11. Latency of routing a packet in a network as packet size increases

The impact of network-scale on the network performance was examined by routing an 8-flit long packet in unloaded networks, as illustrated in Fig.12. The result shows that the asynchronous network caught up its synchronous counterpart as the network scaled up to 4. This is because when the packet size was fixed, the impact resulting from the pipelining due to the recovery operation on the overall latency became constant. As the network scaled up, the latency of header began to dominate and the routing latency of a packet in an asynchronous implementation improved on that of a synchronous implementation.

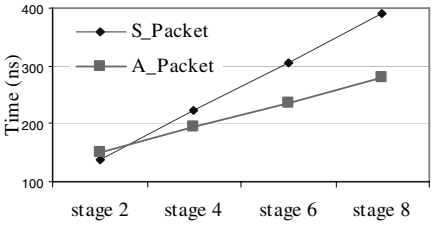

Figure 12. Latency of packet over different numbers of stages

4.3 Gate-counts

The number of gates used in both implementations to achieve the targeted functions is compared. Since the data paths and memory in both implementations share very similar structures, in this paper only the result of the control logic is listed. Both implementations cover a variety of logic gates, these logic is therefore first converted to equivalent standard gates: 2-input NAND. The mapping table is presented in Table 1.

Table 1. Mapping logic to 2-input NAND's

Gate Name	Equivalent Gate-counts	Gate Name	Equivalent Gate-counts
2-Input NAND	1	2-Input AND	1.5
3-Input AND	2	Arbiter	5
D-type flipflop	4	Inverter	0.5
3-Input NAND	1.5	4-Input NAND	2
2-Input NOR	1	3-Input NOR	1.5
3-Input OR	2	4-Input OR	2.5

Table 2. Gate-counts of the asynchronous and the synchronous control logic

Block Name	Asynchronous Implementation		Synchronous Implementation	
	Gate counts	Equiv Gate counts	Gate counts	Equiv Gate counts
(one) Input control logic	161	223.5	73	121.5
(one) Output control logic	83	121	69	128
Total Gate counts	488	689	284	499

Table 2 shows that the asynchronous implementation cost 190 more equivalent gates than the synchronous one. These extra gates mainly came from the input control logic, in which processing header and payload were described into two separate STG's. It nearly doubled the gate-counts of asynchronous input control logic. However, the table shows that the asynchronous output control logic has similar size to its synchronous counterpart.

5. Limitations and future work

It has been known that when contention occurs, metastability in an arbiter for asynchronous circuits may last for an unpredictable period. In this paper, we restrict our performance evaluation and related conclusions to the arbiter's typical behavior only. Secondly, our experiment results were only based on the assumption that the networks were unloaded, and therefore more research should be conducted in investigating the impact of traffic on the performance. Thirdly, a 2-phase signaling protocol should be explored for the improvement of the performance of the asynchronous implementation.

6. Summaries and conclusions

In this paper, we explored the feasibility of implementing an asynchronous on-chip network as the interconnection solution of SoC. In particular, a packet-switch was proposed and implemented. Comparison was made between the asynchronous implementation and its synchronous counterpart. The simulation results suggested that the asynchronous networks could outperform the synchronous networks as the network-scale increased but would underperform the synchronous ones as the packet-size increased. The causes of these phenomena were also explained.

7. References

[1] M.T. Bohr, Interconnect scaling-the real limiter to high performance ULSI, Proc. Int. Electron Devices Meeting, Dec. 1995, pp. 241-244.

[2] L. Benini and G. De Micheli, Networks on chips: a new SoC paradigm, Computer, Volume: 35 Issue: 1, Page(s): 70 - 78, Jan 2002.

[3] L. Benini, G. De Micheli and E. Macii, Designing low-power circuits: practical recipes, IEEE Circuits and Systems Magazine, Vol: 1, Issue: 1, Page(s): 6 -25,2001.

[4] J. Bainbridge and S. Furber, Chain: A Delay-Insensitive Chip Area Interconnect. IEEE Micro 22, 5 Sep. 2002, 16-23.

[5] T.-A. Chu, Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications, PhD Thesis, MIT, June 1987.

[6] D.E. Culler and 1P. Singh, Parallel Computer Architecture, a hardware/software approach, Morgan Kauflnann Publishers, Inc. 1999, USA.

[7] D. Garside, W.J Bainbridge, A. Bardsley, D.M. Clark, DA Edwards, S.B. Furber, 1 Liu, D.W. Lloyd, S. Mohammadi, 1S. Pepper, O. Petlin, S. Temple and 1V. Woods, "AMULET3i-an Asynchronous System-on-Chip", 2001.

[8] P. Guerrier and A, Greiner, A generic architecture for on-chip packet-switched interconnections, Design, Automation and Test in Europe Conference and Exhibition 2000 Proceedings, Page(s): 250 -256,2000.

[9] K Goossens, E Rijpkema, P Wielage, A Peeters and J van Meerbergen, Philips Research, NL, Networks on Silicon: Combining Best-Effort and Guaranteed Services, Design Automation & Test in Europe (DATE) 2002.

[10] Scott Hauck, Asynchronous Design Methodologies: An Overview, Proceedings of the IEEE, Vol.83, No.1, pp69-93, January 1995.

[11] A. Jantsch and H. Tenhunen, Networks on chip, Kluwer Academic Publishers, Hingham, MA, 2003.

[12] M.B. Josephs, S.M. Nowick and c.H. van Berkel, Modelling and Design of Asynchronous circuits, Proceedings of the IEEE on Asynchronous circuits and systems, v. 87:2, Feb., 1999.

[13] M. 1. 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.

[14] A. Lines, Nexus: An Asynchronous Cross-bar Interconnect for Synchronous System-on-Chip Designs, 11th annual Hot Interconnects conference in August, 2003.

[15] F. Thomson Leighton, Introduction to Parallel algorithms and architectures: arrays, trees, hypercubes, Morgan kaufmann Publisher San Mateo, California, 1992.

[16] C. L. Seitz, System Timing. In C.A. Mead and L.A. Conway, editors, Introduction to VLSI Systems, chapter 7. Addison-Wesley, 1980.