Asynchronous Packet-Switch for SoC

Jun Xu, Reza Sotudeh
University of Hertfordshire
x.jun@herts.ac.uk ; r.sotudeh@herts.ac.uk

Abstract:

System-on-Chip (SoC) design is facing increasing difficulties in its integration, global wiring delay and power dissipation. Interconnection network technology has the advantage over the conventional bus technology in its scalability; on the other hand, asynchronous circuit design technology may offer power saving and tackle the clock-skew problem. The combination of these two technologies therefore could be an optimal solution for the interconnection of SoC. In this paper we focus on the implementation of packet-switch with asynchronous technology. The results of experiments run to evaluate several aspects of the packet-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 so far, interconnection networks have been received more and more attentions as an alternative integration solution [2, 7, 8]. One advantage of interconnection networks over buses is the scalability in throughput, latency, cost and integration of the system.

Wire delay is another challenge. The increase in the delay of a global wire, almost doubling every year, affects the signalling, 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. 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 [6].

Power dissipation has become another critical metric. 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 [9]: unlike a synchronous system, charging and discharging (consuming power) in asynchronous circuits takes place only when a circuit is in operation.

Having seen the challenges of 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 detail of the packet-switch by asynchronous technology is presented in Section 3; the simulation results are presented in section 4; finally the conclusions are drawn in Section 5.

2 Architecture of packet-switch

An output-buffering packet-switch [11] is proposed for this study. For ease of our implementation, the packet-switch only consists of two input/output ports as shown in Fig.1. The packet-switch consists of four blocks: Output Block1, Output Block0, Input Block1 and Input Block0. Both input and output blocks comprise of control block and data path. The input data path can buffer one flit (32-bit wide) at a time; the output data path as buffer memory is the main storage element of the packet-switch.

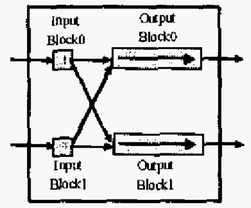

Figure 1 A 2by2 switch

Data transfers between two circuits are based on a Point-to-Point flow control protocol involving requests, which initialise each transfer, and acknowledgements, which signal the completion of a transfer.

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.

Packets at each packet-switch are processed in a pipelined fashion (three stages) as shown in Fig2. Incoming flits from previous packet-switches /source 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 for the packet. If the flit belongs to payload, it is then transmitted to the same memory as its header. In this

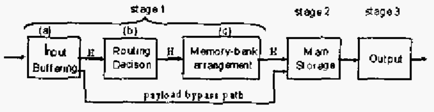

Figure 2 Pipelined processing model

packet-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.

2.1 Input data path

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) for each packet-switch is always the Least Significant Bit (LSB) of each header. This determines the output block with which to communicate.

Making routing decisions is achieved by means of shifting and buffering operations at input data paths. As shown in Fig.3, the input data path includes 33-bit D-type flipflops (DFF's). The extra bit is used to support the shift operation which removes the LSB of each header thus stripping off the used bit of routing information. An incoming flit is transmitted through 32 DFF's, outputting either from bit0 to bit31 or from bit1 to bit32. The former event takes place when the incoming flit is the header and the latter event takes place when it is part of payload. Multiplexers are deployed in front of the DFF's to direct the incoming flit. As a header is buffered at input data paths, the Most Significant Bit (MSB) is filled with "0", bit1 turning into the new LSB which together with the rest of header will be transmitted to its next stage. Both header and payload are read from bit1 to bit32 of input data paths.

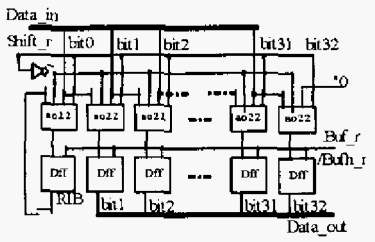

Figure 3 Input data path

2.2 Memory Arrangement

Memory arrangement is mainly conducted by output blocks. The overview of memory arrangement is illustrated in Fig.4, where we assume that the memory consists of two 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 directed 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-1 multiplexers is supplying data.

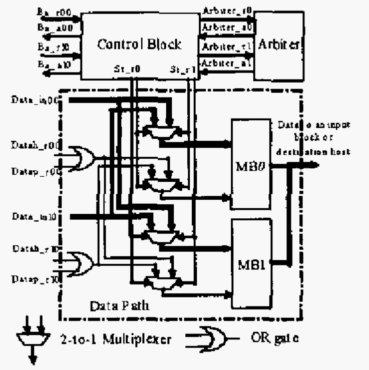

Figure 4 Memory-arrangement at Output Block0

To avoid collision, setting up 2-to-1 multiplexers for different packets must be mutually exclusive: only one action is allowed to progress at a time, therefore, an arbiter 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.

3 Asynchronous implementation

3.1 Asynchronous design methodologies

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 [10]. 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.

3.2 Input control logic

Processing headers and payloads at input blocks is described by two Signal Transition Graphs (STG's) [4], as shown in Fig.5 and Fig.6 respectively.

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.

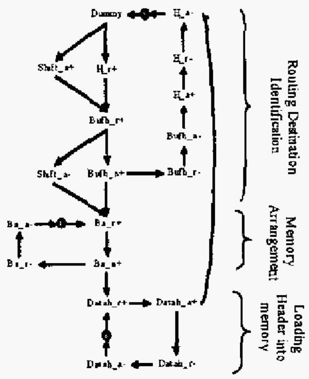
Routing Destination Identification
Memory Arrangement
Loading Header into memory

Figure 5 STG for header-processing at input blocks

Shift_r is the shifting request signal, activated only when an incoming flit is the header, and is released after the header is latched at the input data path. The shifted header is latched 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. The falling transition of *Shift_a* indicates that the routing information has stabilized in the input block.

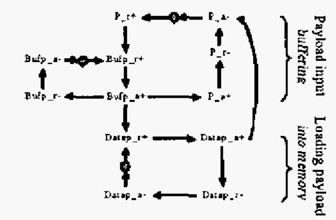

Figure 6 STG for payload-processing at input blocks

Processing (each flit of) payload in an input block is conducted in two steps, i.e., buffering and then transmitting 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* rises as one flit of payload is sent. The input control logic buffers the flit at the input data path by raising *Bufp_r* as soon as *P_r* goes high. *P_a* is driven high as the flit is latched at the input data path, indicated by *Bufp_a* going to high.

3.3 Output control logic

Memory-arrangement in output control blocks is described by an STG presented in Fig.7. *Ba_r00* and *Ba_r10* are the request signals asserted by Input Block0 and Input Block1 for memory-arrangement, and *Ba_a00* and *Ba_a10* are the corresponding acknowledge signals, respectively.

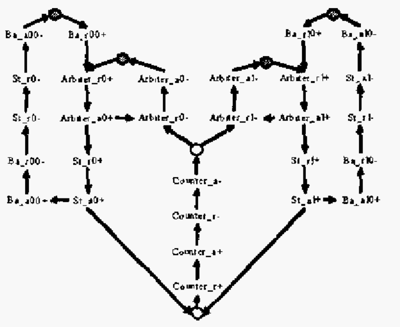

Figure 7 STG for memory-arrangement at Output Block0

Setting up the associated 2-to-1 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 are supplying data, is driven by the handshaking pair *Counter_r* and *Counter_a*. The output control block communicates with the arbiter [13] 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.

Transmitting packets, stored in memory, to their next switches or destination hosts is described in Fig.8. Packets are read out of memory flit by flit using the handshaking pair, *Datao_r* and *Datao_a*, and are forwarded to their next packet-switchs or destination hosts using the handshaking signals, *Inout_r* and *Inout_a*. Note that *Inout_r* and *Inout_a* are the handshaking pair employed between packet-switchs or between a host and a packet-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 the incoming flit is part of the payload. Correspondingly, Signals on *H_a* and *P_a* are multiplexed onto *Inout_a*.

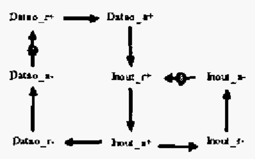

Figure 8 STG for packet-output

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 by Petrify with 0.5μm CMOS technology, and synchronous control circuits were synthesized by SIS. Both implementations were evaluated in MicroSim Design Centre. The minimum clock period, 6ns, was determined by the critical path and obtained from the PSPICE simulation.

The base system used for the simulation was a k-stage butterfly network [12]. The packet size in the evaluation was fixed. For the convenience, the interface between a host and a network was viewed as contributing to the same routing delay as a packet-switch [5].

4.2 Simulation results

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

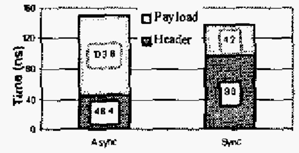

Figure 9 Latency of transmitting an 8-flit packet through a network

The reason that the synchronous packet-switches lost to the asynchronous ones in processing each individual flit was mainly caused by the redundant time in each clock cycle. The 6 ns clock period was dictated by the slowest path as described in Section 4.1. The optimal clock period for these two operations based on our experiment was approximate 4ns. By contrast, the asynchronous circuits immediately progressed as soon as their environment responded.

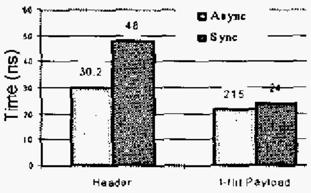

Figure 10 Delay at each switch

When flits are transmitted consecutively in a pipeline style, however, the routing time of each flit can be overlapped by its neighbouring flits. The more they overlap each other, the less routing latency a packet has. For an asynchronous circuit, ruled by a 4-phase level signalling protocol, recovery time was required to return the asynchronous circuit to its original state before another transfer could start. Our simulation result shows the recovery operations caused the asynchronous pipeline less interleaved—only 65.6% of payload-routing time was overlapped, compared to the synchronous network, where 87.5% of overlap rate was

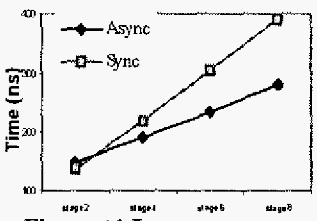

Figure 11 Latency as network scale increases

achieved.

The impact of network scale on its performance is illustrated in Fig.11. The result shows that the performance of the asynchronous networks caught up the synchronous networks after scaling up to 3 stages. It is because in an unloaded network, where the delay of a header at packet-switches is always greater than the delay of payload, the latency of routing a packet is contributed by the time of a header to establish the route from its source to its destination plus the time of loading its payload from its final stage packet-switch to its destination host. The latter is determined by the packet-size, the processing delay at its final stage packet-switch, and the pipeline efficiency; the former is determined by the distance between the source and destination and the delay of header at each packet-switch. When the packet size was fixed, as the network scale (distance) increased, 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.

The impact of packet-size on the performance of networks is presented in Fig.12, where the packet size was varied from 8 flits to 32 flits. The simulation result indicates that increasing packet size can cause longer latency for a packet routing in an asynchronous network than in a synchronous one.

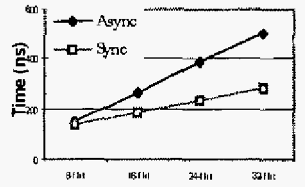

Figure 12 Latency as packet-size increases

4.3 Gate-counts consideration

The gate-counts of synchronous and asynchronous control logic are compared in Table 1. The data paths in both implementations share similar structures, and therefore they are not considered in this paper.

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 1 Gate-counts of asynchronous and synchronous circuits

Table 1 shows that the asynchronous packet-switch has similar size to the synchronous one in output control logic, however, it cost 100 more (equivalent) gates than the synchronous one in input control logic. It is because in the asynchronous input blocks, processing header and payload had to be described using two separate STG's due to the limitation of the asynchronous synthesis tool.

5 Summaries and conclusions

In this paper, we explored the feasibility of an on-chip network using asynchronous circuit design technology as a solution of system integration. A packet-switch was proposed. The asynchronous implementation was presented and compared with its synchronous counterpart. The simulation results suggest that asynchronous networks could outperform synchronous networks as the network-scale increases while underperform with the increase of packet size. The associated reasons were also explained.

6 Acknowledgement

The first author would like to thank UK Overseas Research Students Awards Scheme (ORS) and London South Bank University for their financial support and especially Professor Mark Josephs for his inspiring supervision.

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

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

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

[6] 1D. Garside, W.1 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", http://www.cs.man.ac.uk/amulet/, 2001.

[7] 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.

[8] K Goossens, E Rijpkema, P Wielage, A Peeters and J van Meerbergen, Philips Research, NL, Networks on Silicon: The Next Design Paradigm for System on Silicon, Design Automation & Test in Europe (DATE) 2002.

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

[10] 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.

[11] M. 1. Karol, M. G. Hluchyj, and S. P. Morgan, Input versus output queuing on a space division packet switch, IEEE Transactions on Commtmications, COM-35 (12): 1347-1356, December 1987.

[12] F. Thomson Leighton, Introduction to Parallel algorithms and architectures: arrays, trees, hypercubes, Morgan Kaufinann Publisher San Mateo, California, 1992. [13] G. Moore, VLSI: Some fundamental challenges, IEEE Spectrum, Vol. 16, p.30. 1979.

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