Design and evaluation of a pipelined network interface for network on chip applications

Nguyen Van Cuong, Nguyen Thai Hoang, Pham Ngoc Nam
School of Electronics and Telecommunications
Hanoi University of Science and Technology, Vietnam
Email: {cuong.nguyenvan, nam.phamngoc}@hust.edu.vn

***Abstract*—Recently, Network-on-Chip (NoC) paradigm has been known as a promising solution for complex Systems-on-Chip (SoC) design. A network interface is a significant part of a NoC. The network interface operates as a bridge between computing resources and network routers. This paper introduces a new network interface designed using parallelism in writing and reading buffers processes. The network interface architecture is modeled using Verilog HDL and implemented targeting Xilinx Virtex-6 board. The experimental results proves that this network interface design can guarantee stability as well as 40% improvement in term of latency and throughput compare to the previous design which only use one buffer for both reading and writing processes.**

Keywords—Systems-on-Chip; Network-on-Chip; Network interface; Latency; FPGA.

I. Introduction

In recent years, the technology trends of microchip design and fabrication has reached a very high level of integration. Companies have been enhancing chip fabrication technology to reduce size of transistor on chip as well as increasing working frequency of their chip design. Unfortunately, this design method reached the upper bound of power density. To solve upper bound frequency and power density problems chip designers have decided to utilize multi-core design method instead of frequency scaling. Multi-core systems provide better performance than single core architectures by supporting parallelism in computing. Nowadays, systems-on-chip is used in most embedded system and there are many applications that are integrated on this systems. with application specific, SoC designers often use heterogeneous processors to improve performance instead of using homogeneous processors. According to HiPEAC's vision [1], nowadays communication defines performance. Interconnection networks are of high importance. Due to limitations in bus-based architecture [2,3], SoCs can not provide enough processing power to keep up with this new technology trend.

Network-on-Chip paradigm was proposed and considered as an outstanding alternative for bus-based architecture. NoC architecture is provided a high performance communication infrastructure. Therefore, NoC is a paradigm for integrating a large number of IPs cores for implementing a SoC [4,5], [7]. Main components of a NoC in figure 1 consist of network routers, links, processing resources and network interfaces (NI). A complete network infrastructure that enable resources to communicate and utilize the usage of link is the main function of a NoC [6]. In order to communicate between processing resources and routers, a network interface is required [6]. This network interface has the same function as network interface card that connects personal computer to the internet [6], [9]. The network interface provides services belong to transport layer in the ISO-OSI reference model [8].

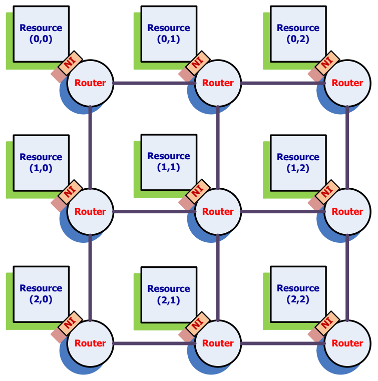


Fig. 1. NoC main components

There are many published researches related to NoC architecture designing recently. In some research, such as [11] the authors presented and implemented a simple network interface architecture for NoC application. However, the NI architecture presented in [11] has a drawback that its latency is high. In research [12], the authors used clock gating technique to optimize the NI for power. Authors in [13,14] have optimized area of the NI so that they can optimize area of the over all NoC. Gray coding was used by authors in [15] to reduce latency in network interface. Ping Pong buffer with four memory modules to improve NI latency was proposed in [16]. Also utilizing memory and reducing latency, research [17] proposed techniques in writing packages mechanism and using standard IP as well as AXI bus to achieve their goals. However, all these research above haven't aimed at reducing latency in reading and writing processing at network interface's FIFO buffers.

This paper proposes a new architecture of network interface that is compatible with 2D Mesh NoC. This network interface is designed to have low latency, high throughput by taking advantage of parallel processing in reading and writing processes. The contributions of this paper:

- Proposed a new architecture for a network interface using 2 FIFO buffers and read/write process parallelism.
- We have successfully implemented our network interface design targeting FPGA Virtex 6 – 6VLX240TFF156 board. Test results show great stabilization and performance.

The remaining part of this paper is organized as following: Section 2 presents our idea and approach, the architecture of our network interface is shown in section 3, section 4 presents experimental results and finally, section 5 contains conclusions and future works.

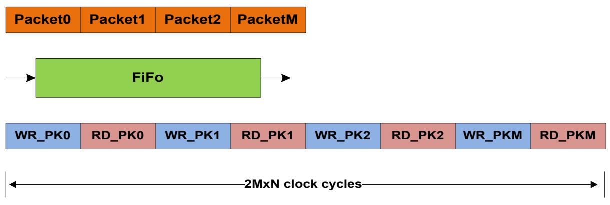
II. The Proposed Approach

A First In First Out (FIFO) buffer is used in the transferring processes of a packet from a router to a data processing resource and vice versa by most author [11,13], [15], [17]. In a FIFO buffer, read and write processes can't happen simultaneously that this process takes place serial, so high latency is inevitable as figure 2a. We think the time a buffer waiting for a packet to be read and be saved by writing the next packet in at the same moment. To realize this idea, we divided 16-bit depth FIFO buffer into two 8-bit depth FIFO buffers and use a flexible algorithms to write and read different packages at the same time as figure 2b. This method was proved to reduce latency significantly.

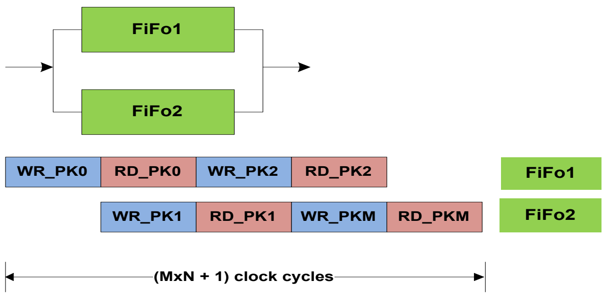
Assume that we want to transfer N packages from a data processing resource to a router or vice versa. Each package will spend M clock cycles.

- Latency (using 1 buffer): 2*N*M (clock cycles)
- Latency (using 2 buffers):

  (2N*M/2) + 1 = (N*M + 1) (clock cycles)
- Latency is reduced by: NxM – 1 (clock cycles)



(a) Using 1 buffer



(b) Using 2 buffers

Fig. 2. Using 1 buffer and 2 buffers

III. Network Interface Architecture Design

A. *Our NoC design Overview*

We has designed a NoC based on 2D Mesh topology using packet switching, wormhole combined with virtual channel flow control, XY routing algorithms. The block diagram of a router in our NoC is shown in Figure 3

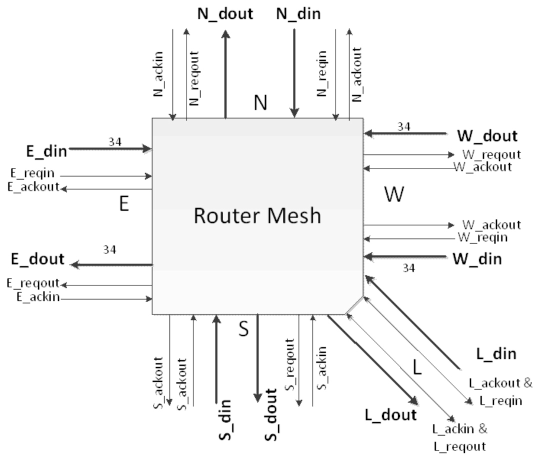
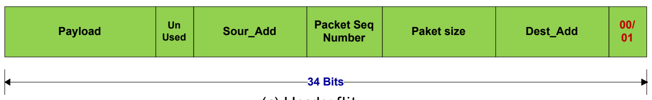
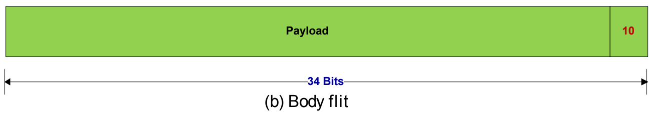


Fig. 3. Router's block diagram

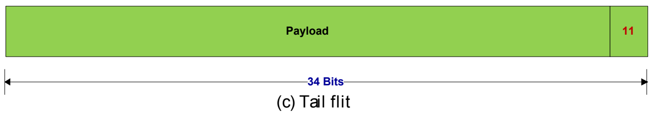
Basic unit in NoC is a flit. There are 3 types of flit: Header flit, body flit and tail flit. Each flit in the network has the width of 34 bits. Flow control information, source address and destination address is contained in header flit. Flits structure is shown in figure 4.



(a) Header flit



(b) Body flit



(c) Tail flit

Fig. 4. Flits' format

B. *Network Interface Architecture*

The network interface is a significant logic block in NoC architecture because its function is to connect data processing resource to the network [18]. A network interface can be divided into two parts: The first part communicates with the network's router. This part is considered as data-independent because all flits have same 34-bit data width as designed. The second part communicates with the data processing resource and based on the resource this part can be changed (data-dependent). Handshake mechanism is used in transferring packages between resource and router.

Detailed block diagram of our network interface is shown in Figure 5. The network interface components consists of: 2 FIFO buffers (A and B), InFSM (C2R and R2C), OutFSM (C2R and R2C), Flitilizer, De-Flitilizer and InstructionFIFO (small size FIFO buffer).

FIFO A and FIFO B are FIFO buffers that are used for storing packages. Each buffer has 8-bit depth and 32-bit width since each flit consists of 32-bit data and 2-bit flit type identifier. By using two FIFOs, we can write and read data from each FIFO simultaneously. When one of these FIFOs is fully loaded with flits, data will be read from this FIFO by the controller of InFSM and OutFSM and pushed to router or resource, and at the same time, the other FIFOs can be used to write new data coming from resource or router. These FIFO buffers is designed as a standard First-In First-Out buffer with read/write enable, full/empty indicators and 32-bit input/output port. During its operation, FIFO's priority is write operation, that is, if both read and write enable signal is asserted at the same time, the write operation will be carried out first as long as the buffer is not full.

InFSM and OutFSM are controlling block that controls data read/write from/to FIFO A and FIFO B. After receiving header flit, packet size and reading/writing mode will be determined by InFSM. There are 4 modes: 01 – Only FIFO A is needed to store the packet, 10 – Only FIFO B is needed to store the packet, 11 – The packet will be written to FIFO A first then FIFO B, 00 – The packet will be written to FIFO B first then FIFO A. This reading/writing mode combined with the packet size is stored in InstructionFIFO as a 8-bit instruction for OutFSM to use later. The OutFSM will read the instructions from InstructionFIFO to perform reading data operation. There are two pairs of InFSM and OutFSM for two route: Core-to-Router (C2R) and Router-to-Core (R2C). C2R InFSM reads the 6-bit packet size in the first flit and then uses status of FIFO A, FIFO B and C2R OutFSM to write a 8-bit instruction to the InstructionFIFO. Instructions stored in InstructionFIFO is read by the C2R OutFSM. Base on the write mode and packet size, OutFSM will read flits from FIFO A and FIFO B in the correct order. When a read operation on a FIFO is completed, C2R OutFSM will clear the FIFO buffer. Operations of R2C InFSM and R2C OutFSM are the same. All transfers to data processing resource or router follow ack/req handshaking mechanism.

Flitilizer and De-Flitilizer are used to attach and detach flit type identifier from the flit. Flit type for the Flitilizer is determined by the OutFSM (C2R). As mentioned above, there are 3 types of flit in a packet and the head flit contains all information used in our routing algorithms. As flits are passed through the Flitilizer, two bits of flit type will be attached to each flit to make 34-bit flits that are used in the network. On the other side, Deflitilizer detaches two bits of flit type from each coming flit to make the original 32-bit flits. Flitilizer and Deflitilizer operate synchronously with the system.

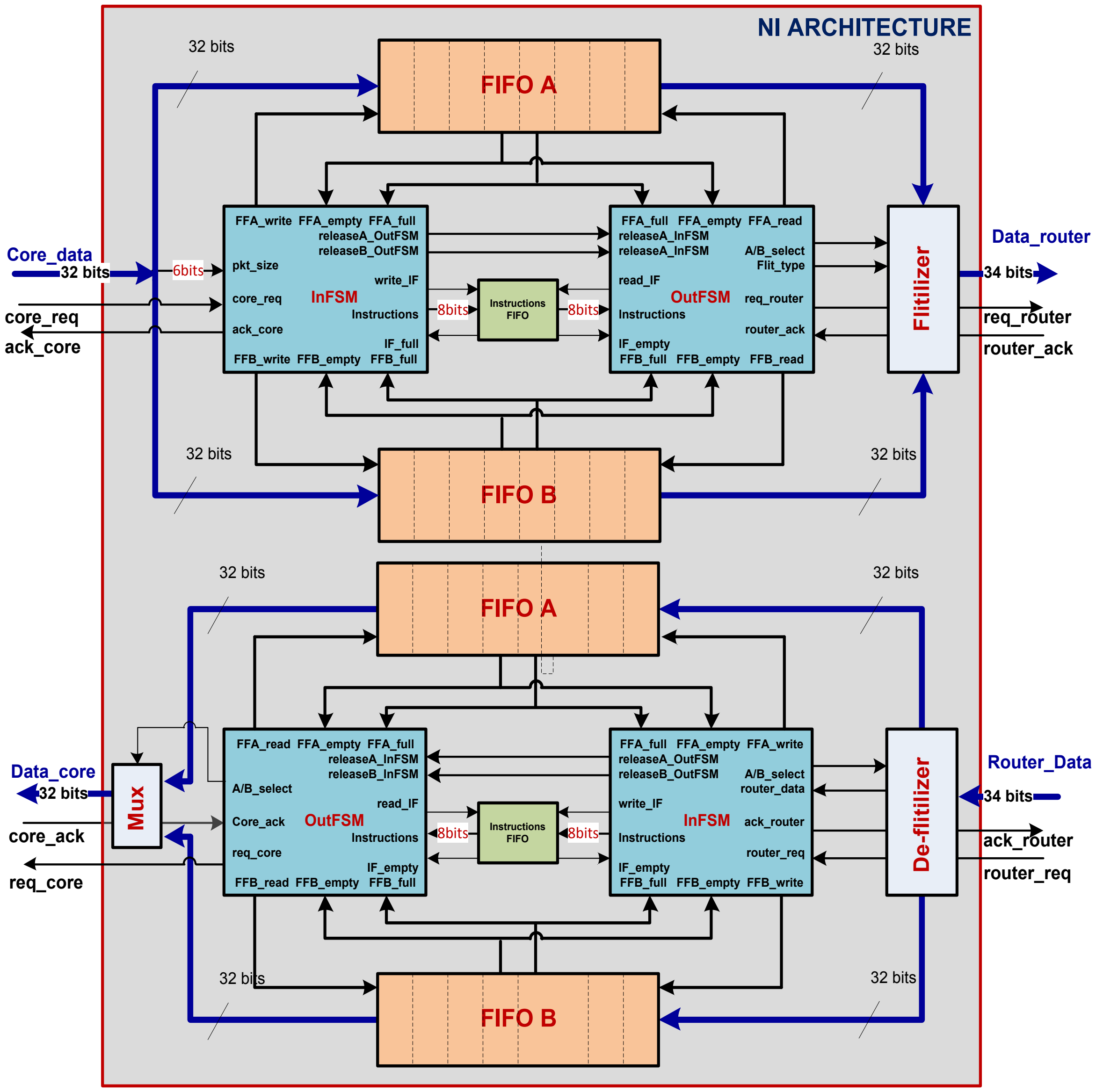


Fig. 5. Detailed block diagram of network interface

IV. EXPERIMENTAL RESULTS

Implementation and simulation results are presented in this section. Furthermore, we also analyzed and evaluated latency as well as throughput of our proposed architecture compare to an architecture that uses one normal FIFO buffer. The network interface is described using Verilog HDL, simulated and implemented targeting Xilinx Virtex 6 ML605 6VLX240TFF156 board by ISE Design Suit 14.1. Our design area utilization is insignificant compare to the device's resources. The synthesis results shown in Table 1.

TABLE I. SYNTHESIS RESULTS FOR VIRTEX 6 ML605 6VLX240TFF156 FPGA

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization (%)**
Slice Registers	316	54567	0.58
Slice LUTs	623	27288	2.28
IOBs	98	296	33.11
No. used as Mem	112	6408	1.75

From simulated and synthesis result, the maximum frequency of a proposed architecture is up to 348 MHz.

Latency and throughput evaluation is conducted by continuously pushing randomly generated packages each with 16 34-bit flits. Random packages is generated by a dummy core and results is tested for correctness by a dummy router that loops the packages back to our NI and data is compared between C2R input and R2C output. The result of the test is recorded and used to measure performance as well as stability of our NI design. The results of our dual buffer design and single buffer design is compared in Table 2 and 3.

TABLE II. THE LATENCY OF NETWORK INTERFACE

Module name	**Latency (Cycle)**		**Gain (%)**
	Our NI	**NI with 1 FIFO**	
FIFO (C2R)	22	32	-30
FIFO (R2C)	22	32	-30
Core – Router	86	113	-31
Router – Core	86	113	-31

TABLE III. THE THROUGHPUT OF NETWORK INTERFACE

Module name	**Throughput @200Mhz (Mbps)**		**Gain (%)**
	Our NI	**NI with 1 FIFO**	
FIFO (C2R)	4200	3200	30
FIFO (R2C)	4200	3200	30
Core – Router	1266	960	30
Router – Core	1190	800	50
NI	**~1228**	**~880**	**+ 40**

Power consumption is estimated by Xpower tool provided by Xilinx ISE Design Suite 14.1. At 200 Mhz, our design power consumption is 1228 mW.

V. CONCLUSION AND FUTURE WORK

In this paper, we has proposed a new architecture of a network interface used in network on chip application. Experimental results show that our design can reduce latency and increase average throughput nearly 40% compare to other method that uses only one buffer. On the other hand, our network interface design consumes an insignificant portion of the Virtex 6 FPGA device resources, therefore, this design is well suited to be integrated in the network on FPGA. In the future, we will continue to improve latency and power of this design by applying clock gating technique and implement as a reconfigurable component on FPGA.

REFERENCES

[1] M. Duranton et al., "The HiPEAC Vision," HiPEAC Roadmap, 2014. [Online]. Available: www.hipeac.net/system/files/hipeacvision.pdf.

[2] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 4th Edition, 4th ed. Morgan Kaufmann, 2006.

[3] J. Liang, S. Swaminathan, and R. Tessier, aSOC: A scalable, single chip communications architecture,in Proc. PACT, 2000.

[4] L. Benini and G. De Micheli,Network on Chips: A New SoC Paradigm,IEEE Computer, Jan.2002, Pages: 70-78.

[5] S. Kumar, ANetwork on Chip Architecture and Design Methodology, Proc. Of IEEE Annual Symposium on VLSI, 2002, Pittsburgh, USA, Pages: 117-124.

[6] Axel Jantsch, Hannu Tenhunen (2004), Networks on Chip, Kluwer Academic Publishers, U.S.

[7] Bashir M. Al-Hashimi (2008), System-on-Chip Next Generation Electronics, the Institution of Engineering and Technology, UK.

[8] M. T. Rose. The Open Book: A Practical Perspective on OSI, Prentice Hall, 1990.

[9] Holsmark R., Johansson A. and Kumar S., "On Connecting Cores to Packet Switched On-Chip Networks: A Case Study with Microblaze Processor Cores", in IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, April 18-21, 2004, Slovakia.

[10] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration,in Proceedings of the 2004 Design, Automation and Test in Europe Conference (DATE'04). IEEE, 2004

[11] W. Chouchene, B. Attia, A. Zitouni, N. Abid, and R. Tourki, R., "A Low Power Network Interface For Network on Chip", in IEEE 8th International Multi-Conference on Systems, Signals & Devices, 2011, pp. 37-42.

[12] B. Attia, W. Chouchene, A. Zitouni, and R. Tourki, "Network interface Sharing for SoCs based NoC", in International Conference on Communications, Computing and Control Applications, 2011, pp. 1-6.

[13] A. Ferrante, S. Medardoni, and D. Bertozzi, "Network Interface Sharing Techniques for Area Optimized NoC Architectures", in DSD, 2008, pp. 10-17.

[14] K.Swaminathan, Lakshminarayanan G and Ko Seok-Bum, "High Speed Generic Network Interface for Network on Chip using Ping Pong Buffers," in International Symposium on Electronic System Design, pp. 72-76, 2012

[15] M. Daneshtalab et all, "Memory-Efficient On-Chip Network With Adaptive Interfaces," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol.31, no.1, pp.146-159, Jan. 2012.

[16] K. Mori et all, "Advanced Design Issue for OASIS Network-on-Chip Architecture," International Conference on Broadband, Wireless Computing, Communication and Applications, 2010.