Efficient Network Interface Architecture for Network-on-Chips

Masoumeh Ebrahimi, Masoud Daneshtalab, N P Sreejesh, Pasi Liljeberg, Hannu Tenhunen
Department of Information Technology, University of Turku, Turku, Finland
{masebr, masdan, srenai, paslil, hanten}@utu.fi

Abstract- In this paper, we present novel network interface architecture for on-chip networks to increase memory parallelism and to improve the resource utilization. The proposed architecture exploits AXI transaction based protocol to be compatible with existing IP cores. Experimental results with synthetic test case demonstrate that the proposed architecture outperforms the conventional architecture in term of latency.

I. Introduction

Network-on-chip (NoC) might be a feasible alternative for the traditional bus-based communication in SoCs due to its reusability, scalability, and parallelism in communication infrastructure [1]. NoCs are composed of routers connecting Processing Elements (PE), to deliver the data (packets) from one place to another [2], and Network Interfaces (NI) the communication interface between each PE and router. The fundamental function of NIs is to provide communication between PEs and network infrastructure. That is, the NI translates the language between the PE and router based on a standard communication protocol such as AXI [3] and OCP [4]. In-order delivery should be handled when exploiting an adaptive routing algorithm for distributing packets through the network [5], or in obtaining memory access parallelization by sending requests from a master IP core to multiple slave memories [6][7]. When a master sends requests to different memories, the responses might be required to return in the same order in which the master issued the addresses, and therefore a reordering mechanism in NoC should be provided by the NI. For implementing an efficient NI, the resource utilization must be improved because in on-chip networks we have limitation of hardware overhead, power consumption, and network latency. According to our observation, the utilization of reorder buffer in NIs is significantly low. Therefore, the traditional buffer management is not efficient enough for NIs. Hence, an advantageous reordering mechanism and resourceful management of buffers in the NI are demanded to increase the utilization and efficiency of the on-chip interconnection network.

The authors in [6] present ideas of transaction ID renaming and distributed soft arbitration in the context of distributed shared memories. This has been improved in [10] by moving reorder buffer resources from NI into network routers. In such systems, because of using global synchronization in the on-chip network, and the delay of release packets recalling data from distributed reordering buffer, the performance might be degraded and the cost of hardware overhead for the on-chip network is too high. Moreover, the proposed architecture is restricted to deterministic routing algorithms. An efficient on-chip NI supporting shared memory abstraction and flexible network configuration is presented by Radulescu et al [7]. The proposed architecture has the advantage of improving reuse of IP cores, and offers ordering messages via channel implementation. Nevertheless, the performance is penalized because of increasing latency, and besides, the packets are routed on the same path in NoC, which forces the routers to use the deterministic routing. Yang et al proposed NISAR [5], a NI architecture using AXI protocol with capability of packet reordering mechanism based on a look up table; but NISAR used a statically partitioned reorder buffer, thereby it had a simple control logic but suffered from low buffer utilization in different traffic patterns. In addition, NISAR does not support the burst transaction, whereas burst type should be handled by the NI. The major contribution of this paper is to propose a novel dynamic buffer allocation architecture for the reorder buffer.

In this work, we introduce new NI architecture providing a novel dynamic buffer allocation based on variable packet size for improving the resource utilization and performance of the NI and on-chip network. The proposed NI architecture enables sharing slots of reorder buffer via a dynamic buffer management [8][9], thereby replacing the conventional, static resource allocation. Besides, the proposed architecture exploits AMBA AXI protocol, to allow backward compatibility with existing IP cores[3]. We also present micro-architectures of the NI, particularly the reordering mechanism to realize the idea. We believe that there are not any documented implementations details of such reordering buffer yet. The paper is organized as follows. In Section II, the proposed architecture is discussed while the results are presented in Section III, and the summary and conclusion given in the last Section.

II. Master and Slave NI Architectures

In the AXI transaction-based model [3][6], IP cores can be classified as master (active) and slave (passive) IP cores [7]. Master IP cores initiate transactions by issuing read and write requests and one or more slaves (memories) receive and execute each request. Subsequently, a response issued by a slave can be either an acknowledgment (corresponding to the write request) or data (corresponding to the read request) [7]. The AXI protocol provides a "transaction ID" field assigned to each transaction. Transactions from the same master IP core, but with different IDs have no ordering restriction while transactions with the same ID must be completed in order. Thus a reordering mechanism in the NI is needed to afford this ordering requirement [3][6][10]. Since IP cores are classified into masters and slaves, the NI is also divided into the master NI (Fig. 1) and slave NI (Fig. 2). Both NIs are partitioned into two paths: forward and reverse. The forward path transmits the AXI transactions received from an IP core to a router; and the reverse path receives the packets from the router and converts them back to AXI transactions. The proposed NIs for both master and slave sides are described in detail as follows.

A. Master-side NI:

As shown in the Fig. 1, the forward path of the master NI transferring requests to the network is composed of an AXI-Queue, a Packetizer unit, and a Reorder unit, while the reverse path, receiving the responses from the network, is composed by a Packet-Queue, a Depacketizer unit, and the Reorder unit. The Reorder unit is a shared module between the forward and reverse paths.

AXI-Queue: the AXI master core transmits write address, write data, or read address to the NI through channels. The AXI-Queue

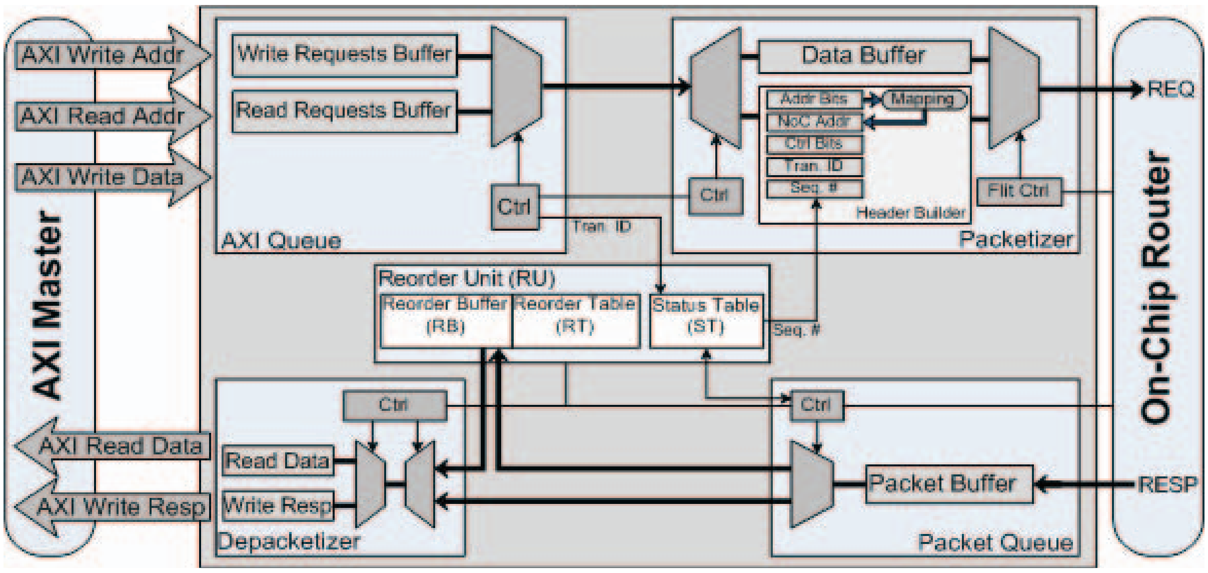

Fig. 1. Master-side NI architecture.

unit performs the arbitration between write and read transaction channels and stores requests into write or read request buffers. The request messages will be sent to the packetizer unit if admitted by the reorder unit, and on top of that a sequence number for each request should be prepared by the reorder unit after the admittance.

Packetizer: it is configured to convert incoming messages from the AXI-Queue unit into header and data flits, and delivers the produced flits to the router. Since a message is composed of several parts, the data is stored in the data buffer and the rest of the message is loaded in corresponding registers of the header builder unit. After the mapping unit converts the AXI address into a network address by using an address decoder, based on the request information loaded on relative registers and the sequence number provided by the reorder buffer, the header of the packet can be formed. Afterward, the flit controller wraps up a packet for convenient transmission.

Packet-Queue: this unit receives packets from the router; and according to the decision of the reorder unit a packet is delivered to the depacketizer unit or reorder buffer. In fact, when a new packet arrives, the sequence number and transaction ID of the packet will be sent to the reorder unit. Based on the decision of the reorder unit, if the packet is out of order, it is transmitted to the reorder buffer, and otherwise it will be delivered to the depacketizer unit directly.

Depacketizer: the main functionality of the Depacketizer unit is to restore packets coming from either the packet queue unit or reorder buffer into the original data format of the AXI master core.

Reorder Unit: it is the most influential part of the NI including a Status-Register, a Status-Table, a Reorder buffer, and a Reorder-Table. In the forward path, preparing the sequence number for corresponding transaction ID, and avoiding overflow of the reorder buffer by the admittance mechanism are provided by this unit. On the other side, in the reverse path, this unit determines where the outstanding packets from the packet queue should be transmitted (reorder buffer or depacketizer), and when the packets in the reorder buffer could be released to the depacketizer unit.

Status-Register and Status-Table: Status-Register is an n-bit register where each bit corresponds to one of the AXI transaction IDs. This register records whether there are one or more messages with the same transaction ID being issued or not. To record the state of the outstanding messages, Status-Table is adopted. Each entry of this table is considered for messages with the same transaction ID, and includes valid tag (v), Transaction ID (T-ID), Number of outstanding Transactions (N-T) as well as the Expecting Sequence number (E-S). The register and table might be updated in both forward and reverse paths described as follows. In the forward path, when the first message of each transaction ID requests for admittance from the reorder unit to enter the network, the corresponding bit in the status register goes high (Fig. 3(a)). The sequence number indicating the order of the messages within the transaction ID is produced by the reorder unit, if the admittance is given. Obviously, this value (Seq-Num) is equal to zero for the first message of each transaction ID. Since only the first message of each transaction does not need to be stored in the reorder buffer, it always gets the admittance to enter the network. If the second request of the same transaction ID is received by the reorder unit, a row in the status table will be initiated for the corresponding message, as demonstrated in Fig. 3(b). In order to prevent overflow of the reorder buffer, if the message is granted by the reorder unit, the required space in the reorder buffer must be reserved according to the size of a new message (size_nm). On top of that, response and request messages might have different sizes (different burst size), therefore, by providing the linked list mechanism in the reorder buffer, the network performance and buffer utilization will be improved considerably. A register, named size_aom, is used to keep the required size of all outstanding transactions in the network. Indeed, this register reserves the number of buffer slots for outstanding messages of different transaction IDs. Since the required buffer space is dependent on the response message size, the required space must be obtained by using the information available in the request message. As shown in Fig. 3(b), before the second message of transaction 2 is admitted by the reorder unit, the reorder unit compares the size_aom value and the reorder buffer size. Then, the message will be admitted if the required space is available in the reorder buffer. While the sequence number of the first message is set to 0, for subsequent messages with the same transaction ID, the sequence number is obtained by adding N-T and E-S values, indicated in Fig. 3(b and c). After the sequence number is obtained, N-T will be increased. E-S indicates the next response sequence number of the corresponding transaction ID to be delivered to the depacketizer unit. In the reverse path, the transaction ID and sequence number of the arriving response packet (message) are sent to the reorder unit to find the related row in the

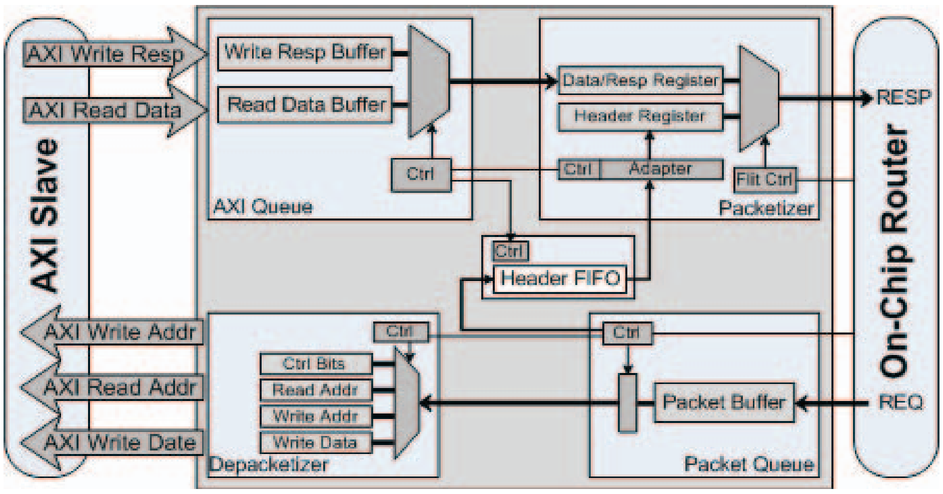

Fig. 2. Slave-side NI architecture.

status table according to the transaction ID (T-ID). If the sequence number of incoming packet is equal to E-S value, the packet is an expected packet (in-order) and should be delivered to the depacketizer unit; thereafter, E-S and N-T values will be increased by +1 and -1, respectively (Fig. 3.d). If N-T value reaches zero, the transaction will be terminated by resetting the valid bit for both status register and status table (Fig. 3.e). However, the packet is out-or-order and should be delivered to the reorder buffer, if the sequence number of the packet is not equal to E-S. Additionally, only one message with the given transaction ID should have been sent to the network, if the given transaction ID is not matched in the status table, thereby only the corresponding bit in the status register will be reset.

Reorder-Table and Reorder-Buffer: As shown in Fig. 4, each row of the reorder table corresponds to an out-of-order packet stored in the reorder buffer. This table includes the valid tag (v), the transaction ID (T-ID), the sequence number (S-N) as well as the head pointer (P). In the reorder buffer, the flits of each packet are maintained by a linked list structure providing high resource efficiency with little hardware overhead. On top of that, the goal of using the shared reorder buffer is to support variable packet size and improve the buffer utilization which can also increase the performance by feeding more packets into the network. Fig. 3 exhibits a pointer field adopted to indicate the next flit position in the reorder buffer. Using the proposed structure in Fig. 4, each out-of-order packet updates the reorder table and reorder buffer. The reordering mechanism guarantees that the reorder buffer will never be overflowed. Therefore, the reorder buffer always has enough space to store the incoming out-of-order packets. Whenever a new packet arrives, some information such as transaction ID and sequence number will be extracted from the header flit and then they will be written into a free slot in the reorder table. Also, the pointer field of the table will be updated to point to a free slot in the reorder buffer. Incoming flits are stored in the reorder buffer cycle by cycle and the pointer field updates to show a next free slot in the reorder buffer. It continues until the tail flit stores in the reorder buffer. Whenever an in-order packet delivered to the depacketizer unit, the depacketizer controller checks the reorder table for the validity of any stored packet with the same transaction ID and next sequence number. If so, the stored packet will be released from the reorder unit to the depacketizer unit.

B. *Slave-side NI:*

A slave IP core cannot operates by itself. It receives requests from master cores and responds to them. Hence, using reordering mechanism in the slave NI is completely meaningless. But to avoid losing the order of header information (transaction ID, sequence number, and etc) carried by arriving requests, a FIFO has been considered. After processing a request in the slave core, the response packet should be created by the packetizer. As can be seen from Fig. 2, to generate the response packet, after the header content of the corresponding request is invoked from the FIFO, and some parameters of the header (destination address, and packet size, and etc) are modified by the adapter, the response packet will be formed. However, the components of slave-side interface in both forward and reverse paths are almost similar to the master-side interface components, except the reorder unit.

III. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

A cycle-accurate 2D NoC simulator is implemented to assess the efficiency of the proposed method. The simulator models all major components of the NoC such as NI, routers, and wires. We use a 25-node (5×5) 2D mesh on-chip network within two different configurations for the entire architecture. In the first configuration (A), out of 25 nodes, ten nodes are assumed to be processor (master cores-with master NI) and other fifteen nodes are memories (slave cores-with slave NI). For the second configuration (B), each node is considered to have a processor and a memory (master core with master-NI, and slave cores with slave-NI). The router has a typical state-of-the-art structure including input buffers, a VC (Virtual Channel) allocator, a routing unit, a switch allocator and a crossbar. Each router has 5 input/output ports, and each input port of the router has 2 VCs. The array size, router algorithm, link width, number of VCs, buffer depth of each VC, and traffic type are the other parameters which must be specified for the simulator.

The routers adopt XY routing and wormhole switching. For all routers, the data width (flit) was set to 32 bits, and the buffer depth of each VC is 5 flits. The baseline architecture (with fixed packet length) uses 1 flit for messages related to read requests and write responses, and 5 flits for data messages, representative of read responses and write requests; the size of read request messages typically depends on the network size and memory capacity of the configured system. As discussed in the previous section, the message size of the proposed mechanism is variable and depends on the request/response length produced by the master/slave core. As the performance metric, we use latency defined as the number of cycles between the initiation of a request operation issued by a

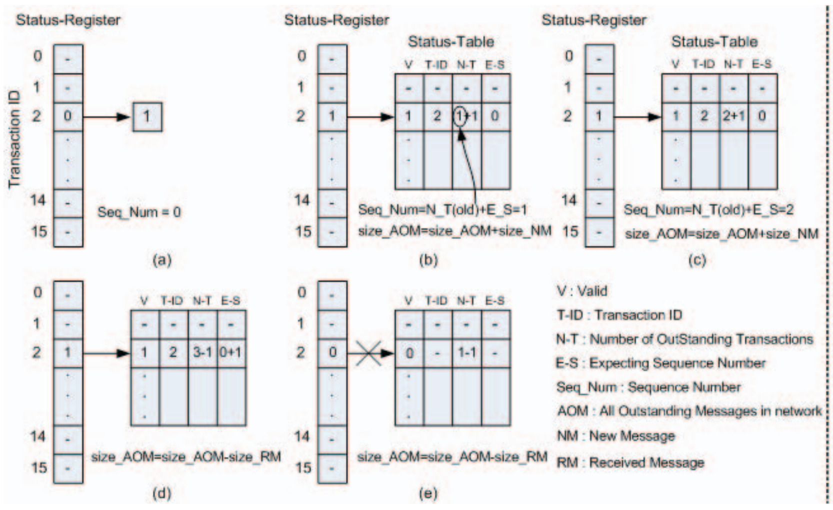

Fig. 3. Updating process of the status register and status table of reorder unit.

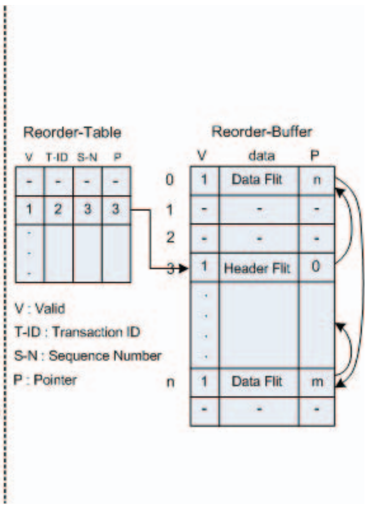

Fig. 4. Dynamic buffer allocation

master and the time when the response is completely delivered to the master from the memory. And the request rate is defined as the ratio of the successful read/write request injections into the NI over the total number of injection attempts. All the cores and routers are assumed to operate at 2GHz. We also set the size of the reorder buffer to 48 words, able to embed 6 outstanding requests with burst size of 8. To evaluate the performance of the proposed schemes, the uniform synthetic traffic pattern has been considered separately for both configurations (A and B). The random traffic represents the most generic case, where each processor sends in-order read/write requests to memories with uniform probability. Hence, the memories and request type (read or write) are selected randomly. Eight burst sizes, among 1 to 8, are stochastically chosen regarding the data length of the request. Fig. 5 reveals that compared with the baseline architecture [5][6] the proposed architecture reduces the average latency when the request rate increases in both of configuration A and B under uniform traffic. One of the foremost reasons of such an improvement is that because the size of packets is not fixed and depends on the request and response lengths, the resource utilization is high and thus, the latency is reduced. Another subtle reason for improving the performance is that getting more free slots in the reorder buffer provides more messages to enter the network. For appraising the area overhead of the proposed architecture, the NI was synthesized by Synopsys D.C. using the UMC 0.09μm technology. In addition to the aforementioned configuration of the NI, the tran_id and seq_id were set to 4-bit and 3-bit respectively. Comparing the area cost of the baseline model for each proposed NI indicates that the hardware overhead of implementing the proposed scheme is less than 0.5%.

IV. Summary and Conclusion

The resource utilization of the conventional reordering methods is not efficient enough; thus, in this work, we presented high performance NI with a novel dynamic buffer allocation which improve the resource utilization, and overall on-chip network performance. Also, the micro-architectures of the proposed master-side and slave-side NIs which are compatible with AMBA AXI protocol have been introduced. A cycle-accurate simulator was used to evaluate the efficiency of the proposed architecture. Under uniform traffic model, in high traffic load, the proposed architectures had lower average communication delay in comparison with the baseline architecture.

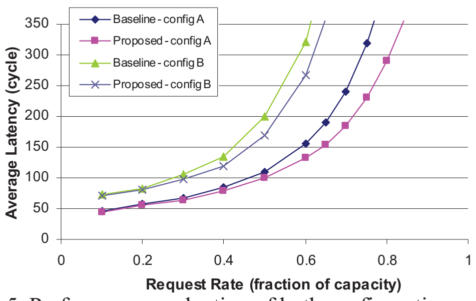

Fig. 5. Performance evaluation of both configurations.

References

[1] B.Towles and W.Dally, "Route packets, not wires: on-chip interconnection networks", Proc. DAC 2001.

[2] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, "RASoC: A Router Soft-Core for Networks-on-Chip", Proceedings of DATE'04, pp. 1530-1591, 2004.

[3] ARM, AMBA AXI Protocol Specification, Mar. 2004.

[4] OCP International Partnership, Open Core Protocol Specification. 2.0 Release Candidate, 2003.

[5] X. Yang, Z. Qing-li, F. Fang-fa, Y. Ming-yan, L. Cheng, "NISAR: An AXI compliant on-chip NI architecture offering transaction reordering processing", in Proc. ASICON, pp. 890-893, 2007, Greece.

[6] W. Kwon, et al., "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories", Proc. DAC, 2008.

[7] A. Radulescu, and et al., "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration", in Proc IEEE TCAD, 24(1), January 2005.

[8] M. H. Neishaburi, Z. Zilic, "Reliability aware NoC router architecture using input channel buffer sharing", in Proc. GLSVLSI, pp. 511-516, 2009.

[9] M. Lai, Z. Wang, L. Gao, H. Lu, K. Dai, "A Dynamically-Allocated Virtual Channel Architecture with Congestion Awareness for On-Chip Routers," in Proceedings of the 46th Design Automation Conference (DAC), pp. 630-633, 2008.

[10] W. Kwon, S. Yoo, J. Um, and S. Jeong, "In-network reorder buffer to improve overall NoC performance while resolving the in-order requirement problem", In proc. DATE'09, pp. 1058 – 1063, France, 2009.