Memory-Efficient On-Chip Network with Adaptive Interfaces

Masoud Daneshtalab, *Student Member, IEEE,* Masoumeh Ebrahimi, *Student Member, IEEE,* Pasi Liljeberg, *Member, IEEE,* Juha Plosila, *Member, IEEE,* and Hannu Tenhunen, *Member, IEEE*

***Abstract*—To achieve higher memory bandwidth in network-based multiprocessor architectures, multiple dynamic random access memories can be accessed simultaneously. In such architectures, not only resource utilization and latency are the critical issues but also a reordering mechanism is required to deliver the response transactions of concurrent memory accesses in-order. In this paper, we present a memory-efficient on-chip network architecture to cope with these issues efficiently. Each node of the network is equipped with a novel network interface (NI) to deal with out-of-order delivery, and a priority-based router to decrease the network latency. The proposed NI exploits a streamlined reordering mechanism to handle the in-order delivery and utilizes the advance extensible interface transaction-based protocol to maintain compatibility with existing intellectual property cores. To improve the memory utilization and reduce the memory latency, an optimized memory controller is integrated in the presented NI. Experimental results with synthetic test cases demonstrate that the proposed on-chip network architecture provides significant improvements in average network latency (16%), average memory access latency (19%), and average memory utilization (22%).**

***Index Terms*—AXI transaction protocol, memory controller, network interface, networks-on-chip.**

I. Introduction

As TECHNOLOGY geometries have shrunk to the deep submicrometer regime, the communication delay and power consumption of global interconnections in high performance multiprocessor systems-on-chip (MPSoCs) are becoming a major bottleneck [1]–[4]. The network-on-chip (NoC) architecture paradigm, based on a modular packet-switched mechanism, can address many of the on-chip communication design issues, such as performance limitations of long interconnects, and integration of high number of processing elements (PEs) on a chip [3]–[5].

NoCs are composed of routers connecting PEs, to deliver the data (packets) from one place to another [6], and network interfaces (NIs) acting as communication interfaces between each PE and corresponding router. The fundamental function of NIs is to provide data transaction between PEs and the network infrastructure. That is, one of the practical approaches of NIs is to translate the protocol between the PE and router based on a standard communication protocol such as advance extensible interface (AXI) [7], open core protocol (OCP) [8], and device transaction level (DTL) [9].

In MPSoCs, in-order delivery is a practical approach which should be handled when exploiting an adaptive routing algorithm for distributing packets through the network [5], [10], when obtaining memory access parallelization by sending requests from a master intellectual property (IP) core to multiple slave memories [11], [12], or when exploiting dynamic memory access scheduling in memory controller to reorder memory requests [13]. In this paper, we present a memory-efficient on-chip network architecture not only to cope with the in-order delivery but also to improve the network performance. The key ideas are threefold.

1) The first idea is to deal with out-of-order handling in such a way that when a master IP-core sends requests to different memories, the responses might be required to return in the same order in which the master issued the addresses. Therefore, we introduce an adaptive NI architecture using a reordering mechanism for the proposed on-chip network. In addition, resource utilization of reorder buffers, implemented in NIs, is significantly inefficient, inasmuch as conventional buffer management is not efficient enough for network resources. Thus, a streamlined adaptive reordering mechanism via resourceful management of buffers is implemented in the NI.
2) As in traditional reordering mechanisms routers do not play any role in the reordering procedure, employing routers in the reordering procedure is very useful to increase utilization and reduce the average delay of on-chip networks. Thus, the second idea is an on-chip router architecture, called priority-based router (PR), which assigns a priority value for each packet according to the sequence number (SN) and distance between source and destination.
3) The third idea is a dynamic memory controller that is integrated into the proposed NI. The presented memory controller is able to reorder memory requests adaptively to improve memory utilization and reduce both memory and network latencies.

Based on the introduced NI architecture, a hybrid NI architecture is designed to integrate both memory and processor in a tile. Furthermore, the presented on-chip network exploits the AMBA AXI protocol to allow backward compatibility with existing IP-cores [7]. We also present micro-architectures of the proposed ideas, particularly the reordering mechanism.

This paper is organized as follows. In Section II, the background is discussed. In Section III, a brief review of related works is presented while the proposed NI architectures are presented in Section IV. The PR and the efficient memory controller are described in Section V and VI, respectively. The experimental results are discussed in Section VII with the summary and conclusion given in the last section.

II. Background

A. NoC Architecture

2-D mesh has many desirable properties for NoCs, including scalability, high bandwidth, and the fixed degree of nodes [14]. A 2-D mesh NoC-based system is shown in Fig. 1. As mentioned earlier, NoC consists of routers (R), PEs, and NI. PEs may be IP blocks or embedded memories. Each core is connected to the corresponding router port using the NI. To be compatible with existing transaction-based IP-cores, we use the AMBA AXI protocol. AMBA AXI is an interfacing protocol, having advanced functions such as a multiple outstanding address function and data interleaving function [7]. AXI, providing such advanced functions, can be implemented on NoCs as an interface protocol between each PE and router to avoid the structural limitations in SoCs due to the bus architecture. The protocol can achieve very high speed of data transmission between PEs [7]. In the AXI transaction-based model [7], [11], IP-cores can be classified as master (active) and slave (passive) IP-cores [12], [24]. 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 a data (corresponding to the read request) [12]. The AXI protocol provides a "transaction ID" (T-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 [7], [8], [18]. The NI lies between a PE and the corresponding attached router. This unit forms the foundation of the generic nature of the architecture as it prevents the PEs from directly interacting with the rest of the network components in the NoC.

A generic NI architecture is shown in Fig. 1. The NI consists of input buffers (forward and reverse directions), a packetizer unit (PU), a depacketizer unit (DU), and a reorder unit (RU). A data burst coming from a PE is latched into the input buffer of the corresponding NI. PU is configured to packetize the burst data stored in the input buffer and transfer the packet to the router. Similarly, data packets coming from the router are latched into the input buffer located in the reverse path. DU is configured to restore original data format, required for the PE, from the packet provided by the router. The RU performs a packet reordering to meet the in-order requirement of each PE.

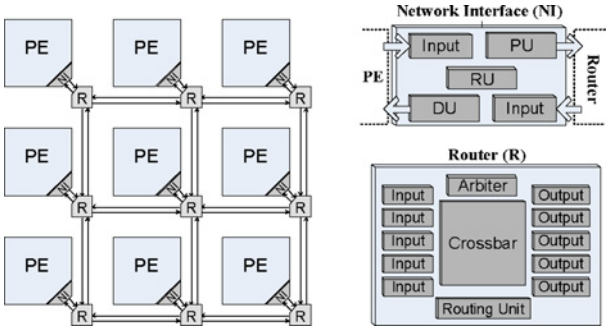


Fig. 1. Tile-based 2-D mesh topology.

As master IP-cores may operate at high clock frequencies and slave IP-cores at low clock frequencies, an interface between the IP-cores and on-chip network is required for crossing two clock domains.

B. DRAM Structure

Dynamic random access memory (DRAM) is designed to provide high memory depth and bandwidth. Fig. 2 shows a simplified 3-D architecture of an DRAM memory chip with the dimensions of bank, row, and column [13], [25], [26]. A DRAM chip is composed of multiple independent memory banks such that memory requests to different banks can be serviced in parallel. Hence, a benefit of a multibank architecture is that commands to different banks can be pipelined. Each bank is formed as a 2-D array of DRAM cells that are accessed an entire row at a time. Thus, a location in the DRAM is identified by an address consisting of bank, row, and column fields. A complete DRAM access may require three commands (transactions) in addition to the data transfer: bank precharge, row activation, and column access (read/write). A bank precharge charges and prepares the bank, while a row-activation command (with the bank and row address) is used to copy all data in the selected row into the row buffer, i.e., sense amplifier. The row buffer serves as a cache to reduce the latency of subsequent accesses to that row. Once a row is in the row buffer, then column commands (read/write) can be issued to read/write data from/into the memory addresses (columns) contained in the row. To prepare the bank for a next row activation after completing the column accesses, the cached row must be written back to the bank memory array by the precharge command [13]. Also, the timing constraints associated with bank precharge, row activation, and column access are t_{RP}, t_{RCD}, and t_{CL}, respectively [13], [25], [27]. Since the latency of a memory request depends on whether the requested row is in the row buffer of a bank or not, a memory request could be a *row hit*, *row conflict*, or *row empty* with different latencies [28]. A *row hit* occurs when a request is accessing a row currently in the row buffer and only a read or a write command is needed. It has the lowest bank access latency (t_{CL}) as only a column access is required. A *row conflict* occurs when the access is to a row different from the one currently

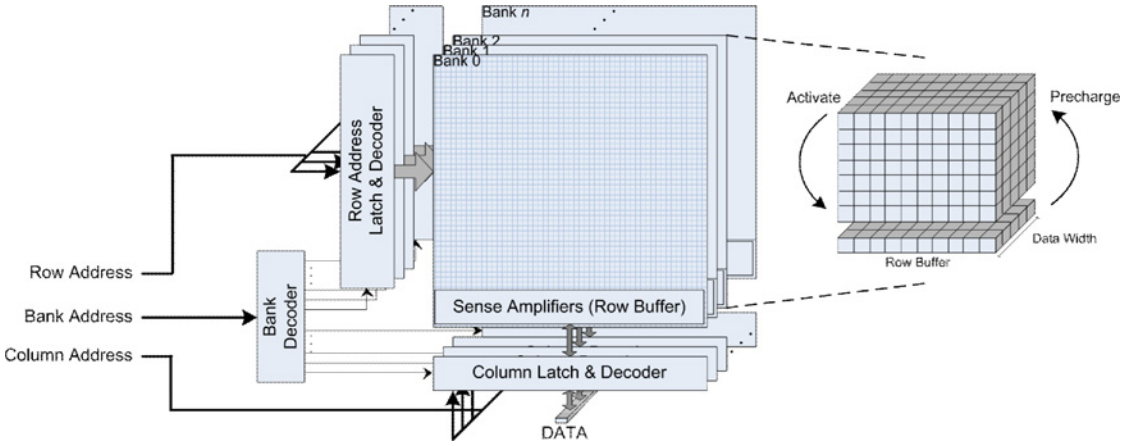


Fig. 2. High-level structure of a DRAM.

in the row buffer. The contents of the row buffer first need to be written back into the memory array using the precharge command. Afterward, the required row should be opened and accessed using the activation and read/write commands. The *row conflict* has the highest bank access latency (t_{RP} + t_{RCD} + t_{CL}). If the bank is closed (precharged) or there is no row in the row buffer then a *row empty* occurs. An activation command should be issued to open the row followed by read or write command(s). The bank access latency in this case is t_{RCD}+t_{CL}.

C. Memory Access Scheduling

The memory controller lies between processors and the DRAM to generate the required commands for each request and to schedule them on the DRAM buses. The memory controller consists of a request table, request buffers, and a memory access scheduler. A request table is used to store the state of each memory request, e.g., valid, address, read/write, header pointer to the data buffer and any additional state necessary for memory scheduling. The data of outstanding requests are stored in read and write buffers. The read and write buffers (request buffers) are implemented as linked lists. Each memory request (read and write) allocates an entry in its respective buffer until the request is completely serviced. Among all pending memory requests, based on the state of the DRAM banks and the timing constraints of the DRAM, the memory scheduler decides which DRAM command should be issued. The average memory access latency can be reduced and the memory bandwidth utilization can be improved if an efficient memory scheduler is employed [13], [25], [26]. Fig. 3 reveals how the memory access scheduling affects the performance. As shown in the figure, the sequence of four memory requests is considered. Requests 1 and 3 are row empties, and requests 2 and 4 are row conflicts. Timing constraints of a DDR2-512 MB used as example throughout this paper are 2-2-2 (t_{RP}-t_{RCD}-t_{CL}) [27]. As depicted in Fig. 3(a), if the controller schedules the memory requests in-order, it will take 22 memory cycles to complete them. In Fig. 3(b), the same four requests are scheduled out-of-order. As can be seen, request 4 is scheduled before requests 2 and 3 to turn request 4 from a row conflict to a row hit. In addition, request 3 is pipelined after request 1, called *bank interleaving*, since it has the different bank address from the bank address of request 1. As a result, only 14 memory cycles are needed to complete the four requests. Thus, how the memory scheduler can improve the memory performance has been shown by this example where the memory utilization of the in-order scheduler and the out-of-order are 4(data)/22(cycle) = 18% and 4/14 = 29%, respectively. In this paper, we present an optimized memory controller that is integrated into the proposed NI to improve the memory utilization and reduce both memory and network latencies. The idea and the implementation details of the proposed architecture are described in Section VI.

III. Related Work

Due to the fact that most of the recently published studies have focused on design and description of NoC architectures, there has been relatively little attention to NI designs particularly when supporting out-of-order mechanisms [24]. The authors in [11] presented ideas of T-ID renaming and distributed soft arbitration in the context of distributed shared memories. In such a system, because of using global synchronization in the on-chip network, the performance might be degraded and the cost of hardware overhead for the on-chip network is too high. In addition, the implementation of ID renaming and reorder buffer can suffer from low resource utilization. This idea has been improved in [18] by moving reorder buffer resources from the NI into network routers. In spite of increasing the resource utilization, the delay of release packets recalling data from distributed reordering buffers can significantly degrade the performance when the size of the network increases [18]. Moreover, the proposed architecture is restricted to deterministic routing algorithms, and thus, it is not a suitable method for adaptive routing. However, neither [11] nor [18] has presented a micro-architecture of the NI. An efficient on-chip NI supporting shared memory abstraction and flexible network configuration is presented by Radulescu *et al.* [12]. 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 the NoC, which forces

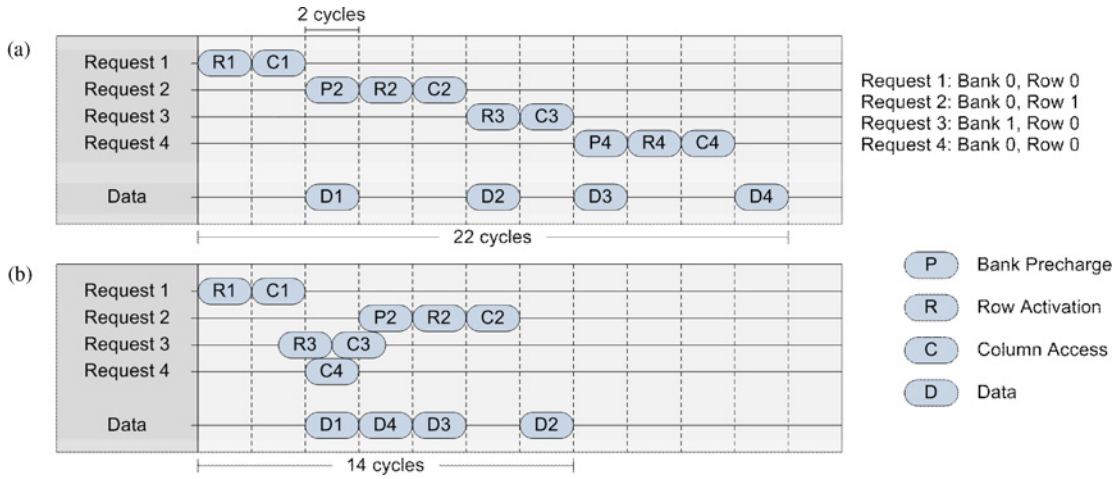


Fig. 3. Memory access scheduling of four memory requests with (a) in-order and (b) with out-of-order access scheduling.

routers to use deterministic routing. Yang *et al.* proposed NISAR [10], a NI architecture using the AXI protocol capable of packet reordering based on a look up table; NISAR has been implemented under the assumption of fixed message size and enjoys simple control logic and design. However, such a mechanism would lead to an inefficient use of network resources for applications that generate periodic variable-size messages (burst mode). Moreover, NISAR suffers from several disadvantages described as follows. First, it permits only a limited number of T-IDs to send several packets to the network so that some requests with different T-IDs are prevented from being serviced for a long period. Second, NISAR uses a statically partitioned reorder buffer suffering from low resource utilization. Third, NISAR presented only a hybrid interface where the master and slave IP-cores are integrated into a single node; however, it imposes significant hardware and delay overhead when the master and slave IP-cores are not integrated into a single node.

In routers, the arbitration process is performed to choose one of multiple input channels to access an output channel. The arbiter could follow either a non-priority or a priority scheme. In the non-priority method, when there are multiple input port requests for the same available output port, the arbiter uses the first-come-first-served [29], also called first-in-first-out (FIFO), or Round-Robin (RR) [30] policy to grant access to an input port. In this way, starvation on a particular port is avoided (fair). On the other hand, in the priority method when there are multiple input port requests for the same available output port, the arbiter would grant access to the input port request which has the highest priority level [31]. The problem with the priority method is that starvation could occur (unfair). In this paper, we introduce a fair PR to improve the network performance with low hardware overhead.

Regarding the memory scheduler, several memory scheduling mechanisms were presented to improve the memory utilization and to reduce the memory latency. The key idea of these mechanisms is in the scheduler for reordering memory accesses. The memory access scheduler proposed in [13] reorders memory accesses to achieve high bandwidth and low average latency. In this scheme, called bank-first scheduling, memory accesses to different banks are issued before those to the same bank. Shao *et al.* [32] proposed the burst scheduling mechanism based on the row-first (RF) scheduling scheme. In this scheme, memory requests that might access the same row within a bank are formed as a group to be issued sequentially, i.e., as a burst. Increasing the row hit rate and maximizing the memory data bus utilization are the major design goals of burst scheduling. The core-aware memory scheduler reveals that it is reasonable to schedule the requests by taking into consideration the source of the requests because the requests from the same source exhibit better locality [26]. In [25] the authors introduced an synchronous dynamic random access memory (SDRAM)-aware router to send one of the competing packets toward an SDRAM using a priority-based arbitration. An adaptive history-based memory scheduler which tracks the access patterns of recently scheduled accesses and selects memory accesses matching the pattern of requests is proposed in [33] and [34]. As NoCs are strongly emerging as a communication platform for chip-multiprocessors, the major limitation of presented memory scheduling mechanisms is that none of them did take the order of the memory requests into consideration. As discussed earlier, requests with the same T-ID from the same master must be completed (returned back) in-order. While requests would be issued out-of-order in memories (slave-sides), the average network latency might be increased significantly due to the out-of-order mechanism in master sides. Therefore, it is necessary to consider the order of memory requests for making an optimal memory scheduling.

The major contribution of this paper is to propose an adaptive NI architecture within a dynamic buffer allocation mechanism for the reorder buffer to increase the utilization and overall performance. That is, using dynamic buffer allocation to get more free slots in the reorder buffer may lead more messages to be entered to the network. On top of that, an efficient memory scheduler mechanism based on the order of requests is introduced and integrated in our NI to diminish both the memory and network latencies. We also present a novel router architecture for incorporating network resources to help in serializing the packets while they progress toward their destinations.

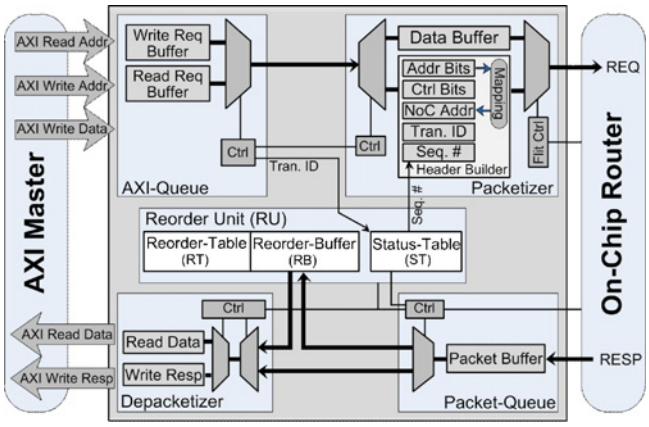


Fig. 4. Master-side NI architecture.

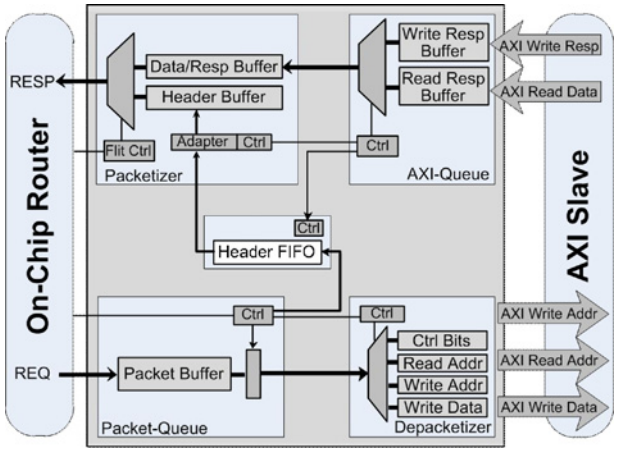


Fig. 5. Slave-side NI architecture.

IV. Proposed NI Architecture

Since IP-cores are classified into masters and slaves, the NI is also divided into the master NI (Fig. 4) and slave NI (Fig. 5). 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 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 Fig. 4, the forward path of the master NI transferring requests to the network is composed of an AXI-queue, a PU, and a RU, while the reverse path, receiving the responses from the network, is composed by a packet-queue, a DU, and the RU. The RU is a shared module between the forward and reverse paths.

1) *AXI-Queue:* The AXI master transmits write address, write data, or read address to the NI through channels. The AXI-queue unit performs the arbitration between write and read transaction channels and stores requests in either write or read request buffer. The request messages are sent to the PU if admitted by the RU, and on top of that a SN for each request should be prepared by the RU after the admittance.

2) *Packetizer:* It converts 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 related registers and the SN provided by the reorder buffer, the header of the packet can be assembled. Afterward, the flit controller wraps up the packet for transmission.

3) *Packet-Queue:* This unit receives packets from the router, and according to the decision of the RU a packet is delivered to the DU or reorder buffer. In fact, when a new packet arrives, the SN and T-ID of the packet are sent to the RU. Based on the decision of the RU, if the packet is out-of-order, it is transmitted to the reorder buffer, and otherwise it is delivered to the DU directly.

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

5) *Reorder Unit:* It is the most influential part of the NI including a status-table, a reorder-buffer, and a reorder-table. In the forward path, preparing the SN for corresponding T-ID and avoiding overflow of the reorder buffer (by an 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 should be released to the DU.

6) *Status-Table:* The state of outstanding messages is kept in a table named status-table. The status-table has n entries where each entry corresponds to a T-ID and n is the number of AXI T-IDs. Each entry contains the information of outstanding messages associated with that T-ID and includes number of outstanding messages (NM), expecting SN (ES), and LMS fields. The NM field reveals how many messages of the given T-ID are inside the network. This value is incremented when a new message with the same T-ID enters the network, and is decremented when the response message comes back to the master core. The ES field points out the SN of the message expected to be delivered to the master core. As the master core expects to receive the first message first, the ES field is set to 0 at the initialization time and it is increased by receiving in-order messages. As already mentioned, each message has a SN indicating the order of the message within the T-ID. This value is produced by the RU, if the message is admitted. Finally, the LMS field defines the reserved buffer space for the last message. The status-table might be updated in both forward and reverse paths described as follows. Suppose that in the forward path, the first message of a T-ID requests to enter the network. The corresponding row of the T-ID is initiated such that the NM, ES, and LMS fields are set to 1, 0, and the message size, respectively, and the value of SN is initialized to 0. However, as no ordering mechanism is required for a single outstanding message of the given T-ID, no buffer space needs to be reserved for this message [Procedure A, Fig. 6(a)].

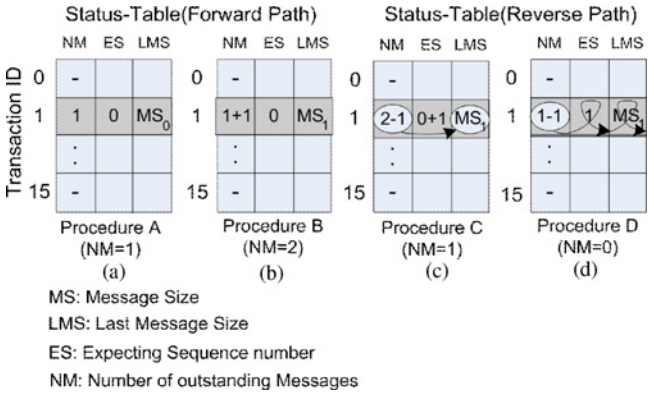


Fig. 6. Status-table of the RU. (a) Procedure A (NM = 1). (b) Procedure B (NM = 2). (c) Procedure C (NM = 1). (d) Procedure D (NM = 0).

For the second (or the rest of) admitted requests of the given T-ID, the NM field is increased by +1, the ES field remains unchanged, and the LMS field is set to the required buffer size of the new message. Subsequently, the value of SN is obtained by adding the values of NM and ES. Since more than one message with the same T-ID is issued (NM $\geq$ 2), the out-of-order handling mechanism is required. Therefore, in order to prevent overflow of the reorder buffer, the buffer space required by the new message is compared with the available space of the reorder buffer. If there is enough space for the new message (*MsgSize*), the required space is allocated in the reorder buffer [Procedure B, Fig. 6(b)]. The *RsrvSize* indicates the required space of all outstanding transactions in the network. Indeed, this register reserves the number of buffer slots required by outstanding messages of different T-IDs.

```
Procedure A:(sending first msg. of T_ID to network)
1 S_Table(T_ID)(NM)      <= "0001";
2 S_Table(T_ID)(ES)      <= (others =>'0');
3 S_Table(T_ID)(LMS)     <= MsgSize;
4 SN                     <= (others =>'0');
5 RsrvSize               <= RsrvSize;

Procedure B:(sending other msgs. of T_ID to network)
1 S_Table(T_ID)(NM)      <= S_Table(T_ID)(NM) + 1;
2 S_Table(T_ID)(ES)      <= S_Table(T_ID)(ES);
3 S_Table(T_ID)(LMS)     <= MsgSize;
4 SN                     <= S_Table(T_ID)(NM) +
                            S_Table(T_ID)(ES);
5 RsrvSize               <= RsrvSize + MsgSize;
```

In the reverse path, the T-ID and SN of the arriving response message are sent to the RU to find the related row in the status-table. In the corresponding row of the T-ID, if the SN of the incoming packet is equal to the value of ES, the packet is an expected packet (in-order) and it should be delivered to the DU. Thereafter, the received message size (*RecvMsgSize*) is reduced from the *RsrvSize,* and the values of ES and NM are added by +1 and −1, respectively [Procedure C, Fig. 6(c)]. However, if the SN of the packet is not equal to the value of ES, the packet is out-of-order and should be delivered to the reorder buffer. In case that the message is delivered to the DU and the value of NM becomes 1, the reserved buffer space for the last message (i.e., LMS) can be deallocated. If the value of NM reaches 0, the transaction is terminated [Procedure D, Fig. 6(d)].

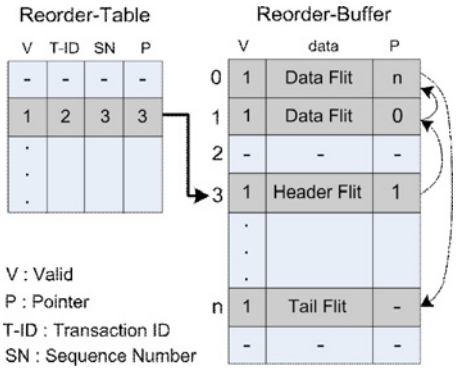


Fig. 7. Dynamic buffer allocation.

```
Procedure C: (arriving expected packet)
1 S_Table(T_ID)(NM)      <= S_Table(T_ID)(NM) − 1;
2 S_Table(T_ID)(ES)      <= S_Table(T_ID)(ES) + 1;
3 S_Table(T_ID)(LMS)     <= S_Table(T_ID)(LMS);
4 RsrvSize               <= IF NM/=1 THEN RsrvSize –
                            RecvMsgSize; ELSE RsrvSize –
                            RecvMsgSize - LMS;

Procedure D: (arriving last packet)
1 S_Table(T_ID)(NM)      <= S_Table(T_ID)(NM) − 1;
2 S_Table(T_ID)(ES)      <= (others =>'0');
3 S_Table(T_ID)(LMS)     <= (others =>'0');
4 RsrvSize               <= RsrvSize;
```

7) *Reorder-Table and Reorder-Buffer:* As shown in Fig. 7, 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 T-ID, the SN 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 a little hardware overhead. On top of that, the goal of using the shared reorder-buffer is to support variable packet sizes and improve the buffer utilization which can also increase the performance by feeding more packets into the network. Fig. 7 exhibits a pointer field adopted to indicate the next flit position in the reorder-buffer. Using the proposed structure in Fig. 7, each out-of-order packet updates the reorder-table and reorder-buffer according to the Procedures E and F.

```
Procedure E: (updating Reorder-Table)
1 ReorderTable(FreeRow)(V)       <= '1';
2 ReorderTable(FreeRow)(T_ID)    <= HeaderFlit(T_ID);
3 ReorderTable(FreeRow)(SN)      <= HeaderFlit(SN);
4 ReorderTable(FreeRow)(P)       <= Current_Free_Slot;

Procedure F: (updating Reorder-Buffer)
1 ReorderBuf(Current_Free_Slot)(V)      <= '1';
2 ReorderBuf(Current_Free_Slot)(Data)   <= flit;
3 ReorderBuf(Current_Free_Slot)(P)      <=Next_Free_Slot;
4 Current_Free_Slot                     <=Next_Free_Slot;
```

The first three operations in the Procedure E store the T-ID and SN from the header flit of the out-of-order packet to the available slot indicated by *FreeRow* in the reorder-table, and the last operation in the Procedure E updates the pointer to point to the available slot in the reorder-buffer. The Procedure F is intended to store the incoming flits into the reorder-buffer. While *Current_Free_Slot* shows the current free location in the reorder-buffer to store the current flit, *Next_Free_Slot* returns an available slot for the next flit. By repeating the

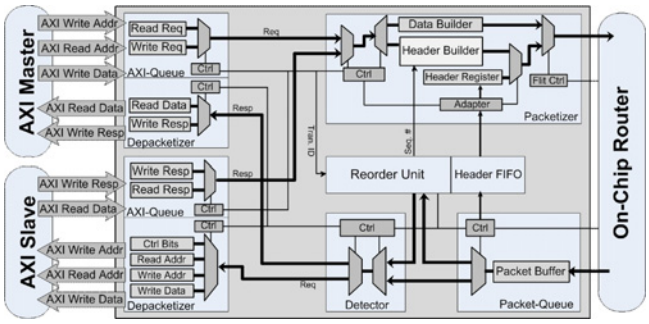


Fig. 8. Hybrid NI architecture.

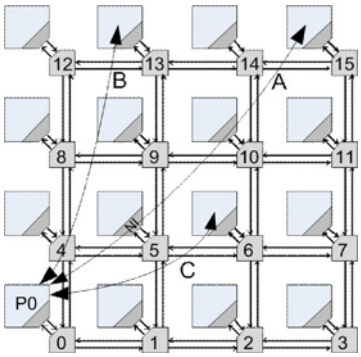


Fig. 9. 4×4 NoC where master core 0 sends requests A, B, and C to memories 6, 13, and 15, respectively.

operations in the Procedure F, all the payload flits are stored in the reorder-buffer. The tail flit can be determined by extracting header flit information. Whenever an in-order packet delivered to the DU, depacketization checks the reorder-table for the validity of any stored packet with the same T-ID and next SN. If so, the stored packet(s) is (are) released from the RU to the DU. If master cores, slave cores, and the network operate at different frequencies, bi-synchronous FIFOs are deployed between NIs and cores. Bi-synchronous FIFOs are widely used in multi-clock systems to synchronize signals from different clock/frequency domains. Each domain is synchronous to its own clock signal but can be asynchronous with respect to others in either clock frequency or phase [19]. The challenges of designing bi-synchronous FIFOs include the enhancement of reliability and reducing latency and power/area cost. We identify the bi-synchronous FIFOs structure presented in [20] can be used in the interfaces.

B. Slave-Side NI

A slave IP-core cannot operate independently. It receives requests from master cores and responds to them. Hence, it is not required to use reordering mechanism in the slave NI. To avoid losing the order of header information (T-ID, SN, and so on) 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. 5, 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, packet size, and so on) are modified by the adapter, the response packet can be formed. However, the components of slave-side interface in both forward and reverse paths are similar to the master-side interface components, except the RU.

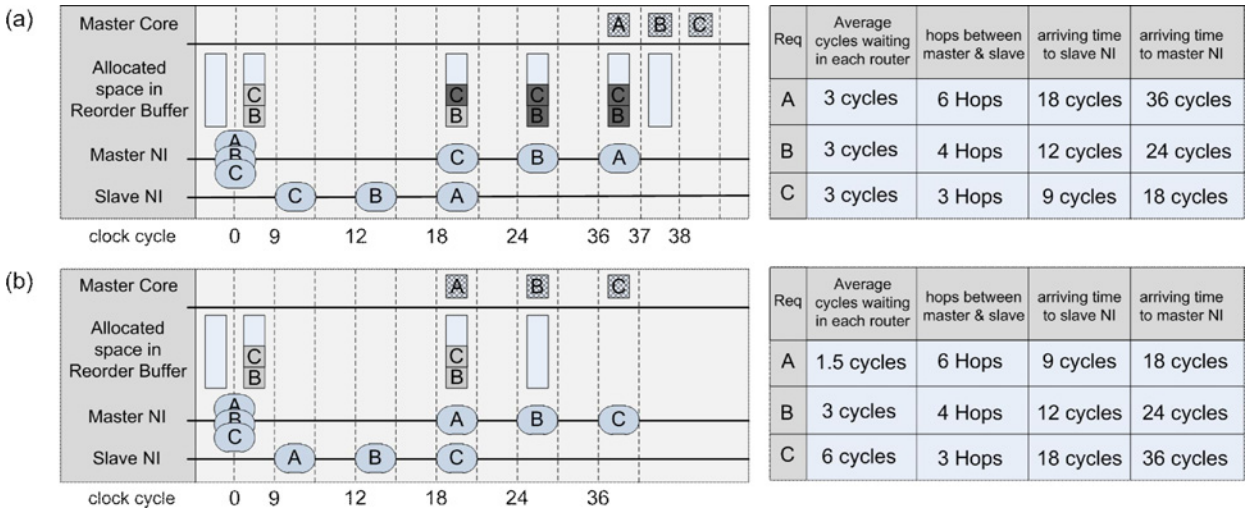
C. Hybrid NI

The hybrid model is formed by combining the *master-side* and *slave-side* NIs. As illustrated in Fig. 8, based on the type of incoming packet (Req/Resp), the detector unit determines the target unit (slave-side queue/master-side queue). Regarding the MPSoC's configuration, if each node is supposed to integrate a dedicated processor and memory, instead of using two NIs (master and slave), the hybrid model is more beneficial, particularly in terms of area and power costs. This architecture also prevents the local requests from entering the network such that local requests can access the local memory directly. A RR arbitration scheme is used between the local requests and global requests coming from the network.

V. PR Architecture

A. Basic Idea of PR

In this part, we present a novel method for incorporating network resources in serializing the packets while they traverse inside the network. Fig. 9 shows a 4×4 tile architecture where the master core 0 accesses three memory modules 6, 13, and 15. Assume that the master core generates three requests, A, B, and C, with a same T-ID and sends request A to memory 15, request B to memory 13, and request C to memory 6. Due to the in-order requirement of the AXI protocol, response A needs to be delivered to the master core first and then responses B and C, respectively. For simplicity, we assume that the memory modules return responses with zero latency and we also assume that a RR arbiter is used in each router such that on average three cycles are needed for a packet to win arbitration in a router. As shown in Fig. 10(a), the master NI sends requests A, B, and C at time 0 to the network. According to the proposed NI architecture, buffer space should be reserved for requests B and C. By considering three cycles waiting time at each router, requests C and B access the memory modules 6 and 13, respectively, at cycles 9 and 12. At cycle 18, request A accesses the memory 15 and meanwhile the response C reaches the master NI. Response C cannot be served by the master core before responses A and B, so it has to be stored in the reorder buffer. At cycle 24, response B is received by the master NI and stored in the reorder buffer as it cannot be served earlier than response A. Response A is received by the master NI at time 36 and it can be sent directly to the master core. Finally, the stored responses B and C are released from the reorder buffer and delivered to the master core at cycles 37 and 38, respectively. However, when response B is delivered to the master core, the allocated buffer space for both requests B and C is released. The idea behind our method is to give better chance to the long-distance packets with low ordering values to win arbitration in routers. By this approach, in the same example as Fig. 10(a), the number of waiting times of request A in arbitration phases is probably less than that of requests B and C (similarly, the number of waiting cycles



Req	Average cycles waiting in each router	hops between master & slave	arriving time to slave NI	arriving time to master NI
A	3 cycles	6 Hops	18 cycles	36 cycles
B	3 cycles	4 Hops	12 cycles	24 cycles
C	3 cycles	3 Hops	9 cycles	18 cycles

Req	Average cycles waiting in each router	hops between master & slave	arriving time to slave NI	arriving time to master NI
A	1.5 cycles	6 Hops	9 cycles	18 cycles
B	3 cycles	4 Hops	12 cycles	24 cycles
C	6 cycles	3 Hops	18 cycles	36 cycles

Fig. 10. Comparing (a) RR and (b) priority-based arbitration schemes in serializing the packets.

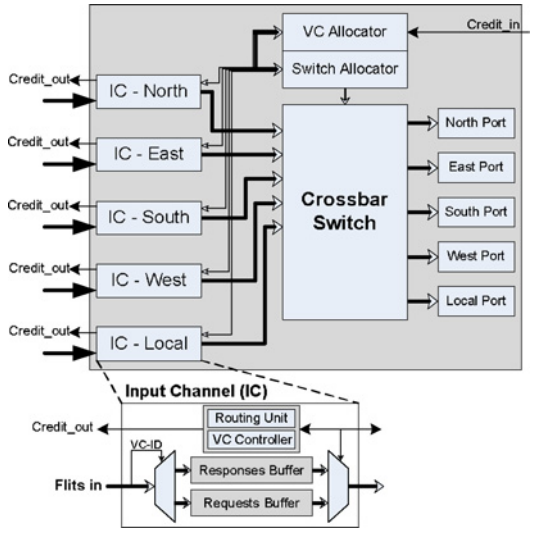


Fig. 11. Router architecture.

```
i    : i(th) input channel
P(i) : priority value of i(th) input channel
       = (MaxSeqNum - PacketSeqNum + Distance)
WP(i): waiting periods + P(i)
---------------
process Find_MaxPriority is
begin
  MaxValue <= 0;
  for 'i=0 to all Reqs in the output port' loop
     if Req is a new packet then
        WP(i) <= P(i);
     else
        WP(i) <= WP(i) + 1;
     end if;
     if WP(i) > MaxValue then
       MaxValue <= WP(i);
       select   <= i;
     end if;
  end loop;
end process;
```

Fig. 12. Pseudo VHDL code of the PR.

of request B is probably less than request C). The possibility of benefits from the idea of PR can be found in the case of Fig. 10(b) in which the requests experience different waiting periods at routers and they are supposed to be 1.5, 3, and 6 cycles for requests A, B, and C, respectively [these values are chosen such that the results could be compared with the example shown in Fig. 10(a)].

As illustrated in Fig. 10(b), requests A and B access their memories at cycles 9 and 12, respectively. At cycle 18, request C accesses the memory while response A is received by the master NI. Since response A arrived in-order, it can be served immediately and delivered to the master core. At cycle 24, response B can also be directly sent to the master core. Upon arrival of this response not only the required space for request B is released but also the reserved buffer space for request C is released. Finally, the response C reaches the master NI at cycle 36 and it is delivered to the master core directly. According to the examples in Fig. 10(a) and (b), latencies can be considerably reduced by applying the idea of PR, i.e., in Fig. 10(b), responses A, B, and C are delivered to the master core at cycles 18, 24, and 36 which are earlier than in Fig. 10(a), where responses are served at cycles 36–38, respectively. This reduction is mainly from the fact that responses arrive at the master NI in-order, and thus can be served immediately. Another advantage of using PR is that the corresponding reserved buffer space in the reorder buffer can be released sooner as the responses are mainly reaching the master NI in-order, i.e., in Fig. 10(b), the reserved buffer space for requests B and C are released at cycle 24 while in Fig. 10(a) they are freed at cycle 37. The idea of PR can further improve the performance by allowing more pending requests to enter the network, thereby reducing the overall latencies of packets.

B. Proposed PR

The architecture of the PR, depicted in Fig. 11, has a typical state-of-the-art structure including input buffers, a virtual channel (VC) allocator, a routing unit, a switch allocator, and a crossbar. Each router has five input/output ports, and each input port of the router has two VCs. Packets of different

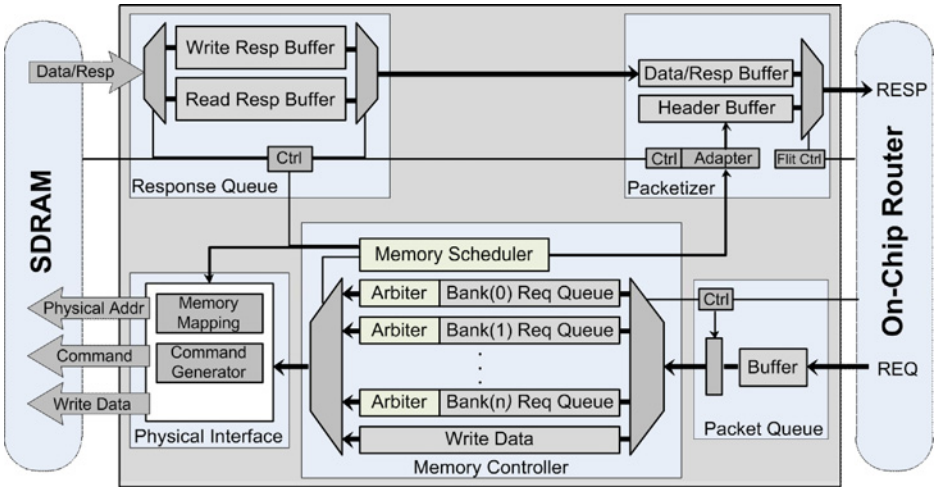


Fig. 13. Proposed memory controller integrated in the slave-side NI.

message types (request and response) are assigned to corresponding VCs to avoid message dependency deadlock [21]. The arbitration scheme of the switch allocator in the typical router (TR) structure is RR. The RR scheme is a fair policy when all packets have the same priority, otherwise priority-based methods are more beneficial. As already mentioned, each packet is assigned a SN and packets might be returned back out-of-order due to the different path length and different memory response time. The PR assigns a priority to each packet such that long-distance packets with low SNs have better chance to win the arbitration in routers. Accordingly, the packet priority is computed by summing up two values. The first one is the distance the packet must traverse between the master and slave while the second one is obtained by subtracting the *MaxSeqNum* value (the maximum SN value that can be generated by the NI) from the *PacketSeqNum* value (packet SN). The result is stored in the packet's header and used by router arbiters. By using the PR, packets can proceed inside the network with different speeds according to their priority values. Fig. 12 shows the algorithm in which the process, *Find_MaxPriority*, is activated when the output channel is available and there are multiple messages. It examines all messages and the priority value of the corresponding input packets and grants a message with the highest priority value. In order to prevent starvation, each time after finding the highest value, the priorities of defeated packets are incremented.

VI. Order Sensitive Memory Scheduler

The architecture of the proposed memory controller, dubbed OS from order sensitive, is depicted in Fig. 13. As illustrated in the figure, the proposed memory controller is integrated in the slave-side NI. After arriving at the NI on the edge of the network, requests are stored in the respective queues based on their target banks. The data associated with write requests is stored in the write queue. The queues are implemented as the linked list structure which has been described in Section IV. Depending on the SN, received requests in each bank queue obtain a priority value to access the memory. Here we have the same scenario as in the PR. The priority value of each packet is based on the SN, which helps to deliver the packets to the master NI in-order such that the corresponding reserved buffer space in the reorder buffer can be released sooner. Once a new request enters a queue, the process *input_queue*, shown in Fig. 14, updates the priority value of each request in the queue. The packet's SN of received request is assigned as a priority value for this request. In addition, to prevent starvation, the priority values of existing requests in the queue are incremented at every *input_queue* event. As mentioned earlier, each bank arbiter selects a request from the queue with the highest priority value based on the bank timing constraints as the first level of scheduling procedure. Since the RF policy has better memory utilization in comparison with the other bank arbitration policies, the bank arbiters of the presented memory controller also takes advantage of the RF policy. The bank arbitration policy in our memory controller is shown in Fig. 14. Whenever the *arbiter* process is activated, it tries to find a request which is a row hit and has a higher priority value. If there are not any row hits, the bank arbiter selects the highest priority request which is a row conflict from the queue and issues the DRAM commands to service the selected request. Fig. 15 depicts the circuit of finding the suitable request in the request buffer which is described in the *arbiter* process of Fig. 14. In the second level of the scheduling procedure, at each memory cycle the memory scheduler decides which request from all bank arbiters should be issued. To simplify the hardware implementation and provide the *bank interleaving*, RR mechanism is utilized by the memory scheduler.

VII. Experimental Results

In this section, we evaluate the proposed on-chip network architecture in terms of average network latency, memory latency, and memory utilization compared with the baseline architecture under different traffic patterns. Also, we discuss the area and power consumption of the proposed NoC components: NI, PR, and OS memory scheduler. Consequently, a 2-D NoC simulator is implemented with VHSIC hardware description language (VHDL) to model all major components of the NoC.

```
RA(i) : row address of i(th) request.
CRA   : current row address issued prior.
P(i)  : priority value of i(th) request
        =(MaxSeqNum - PacketSeqNum)
W(i)  : waiting periods + prioriy
        of i(th) request in the queue.
----------------
Process(input_queue)
Begin
For 'i:1 to number of Reqs in input queue' loop
    If Req is a new packet then
       W(i) <= P(i);
    Else
       W(i) <= W(i)+1;
    End if;
 End loop;
End process;
----------------
Process(arbiter)
Begin
 MaxValue1 <=0; select1 <=0;
 MaxValue2 <=0; select2 <=0;
   For 'i:1 to all requests in input queue' loop
     If RA(i)= CRA then
       If W(i)>= MaxValue1 then
         select1     <= i;
         MaxValue1   <= W(i);
       End if;
     Else
       If W(i)>= MaxValue2 then
         select2     <= i;
         MaxValue2   <= W(i);
       End if;
    End if;
   End loop;
   If select1 /= 0 then
       select <= select1;
   Else
       select <= select2;
   End if;
End process;
```

Fig. 14. Pseudo VHDL code of the arbiter in the memory controller.

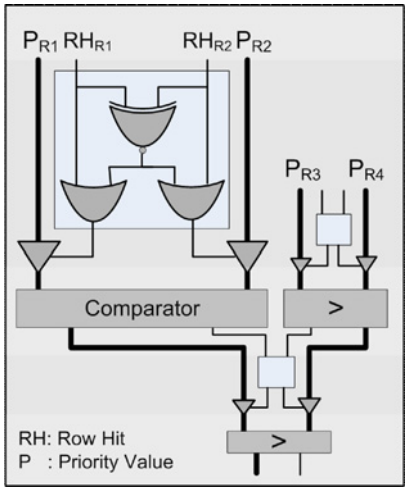


Fig. 15. Request selector circuit.

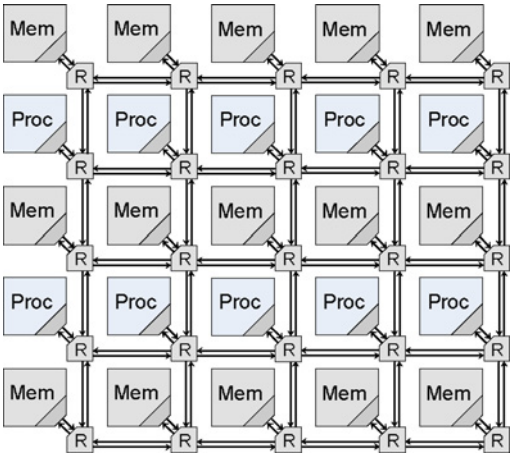


Fig. 16. Layout of the system configuration A.

A. System Configuration

In this paper, we use a 25-node (5×5) 2-D mesh on-chip network with two different configurations for the entire architecture. In the first configuration (A), illustrated in Fig. 16, out of 25 nodes, 10 nodes are assumed to be processors (master cores, connected by master NIs) and remaining 15 nodes are memories (slave cores, connected by slave NIs). For the second configuration (B), each node is considered to have a processor and a memory (master and slave cores, connected by a hybrid NI). The processors are 32b-AXI and the memories specified in Section II, are DDR2-256 MB (t_{RP}-t_{RCD}-t_{CL} = 2-2-2, 32b, 4 banks) [27]. We assume that the memories are integrated on a separate die which is stacked on top of the processor layer [35]–[37]. Inasmuch as the memories are now stacked on top of the processors layer, the front-side bus and memory controller operate at the same speed as the processors. The timing of each stacked DRAM module is still the same as in a traditional DRAM memory (t_{CAS}, t_{RAS}, and so on, are unchanged) [35]–[37]. We adopt a commercial memory controller with memory interface, DDR2SPA module from Gaisler IP-cores [38]. Along with the proposed OS memory scheduler, another memory scheduler with RF policy is also implemented as the default scheduler for the memory controller. The network of each configuration that has been considered for experimental results is formed either by TR or by PRs.

The array size, routing 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 the XY routing algorithm [22], [23] and utilize wormhole switching. For all routers, the data width (flit size) is set to 32 bits, and the buffer depth of each VC to five flits. Message structures for the AXI protocol are defined in [12] and [28]. For the request, the command and all its control bits (flags) are included in the first flit of the packet, the memory address is set in the second flit, and the write data (in the case of a write command) is appended at the end. For the response message, the control bits are included in the first flit while the read data is appended at the end if the response relates to a read request. Hence, the packet length for write responses and read requests are 1 flit and 2 flits, respectively, while the packet length for data messages, representative of read responses and write requests, is variable and depends on the write request/read response length (burst size) produced by a master/slave core. As a performance metric, we use latency defined as the number of cycles between the initiation of a request operation issued by a master (processor) and the time when the response is completely delivered to the master from the slave (memory). The request rate is defined as the ratio of

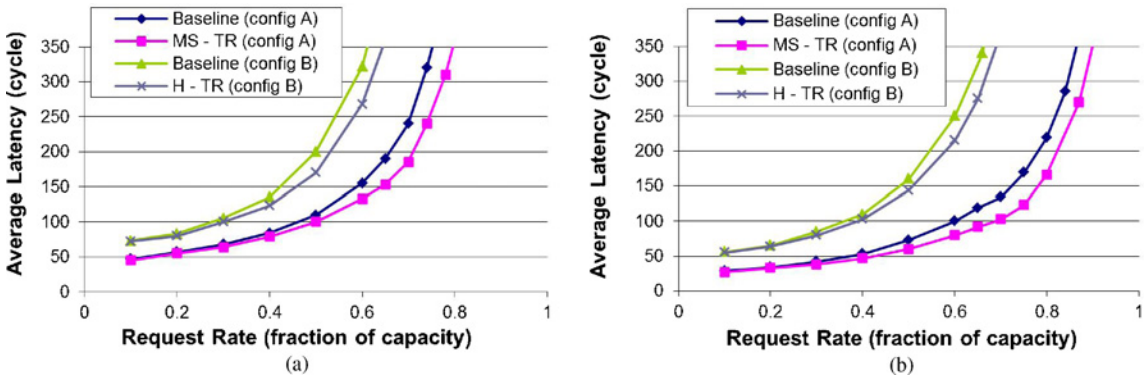


Fig. 17. Performance evaluation of both configurations under (a) uniform and (b) non-uniform traffic models.

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 1 GHz, and for fair comparison, we keep the bisection bandwidth constant in all configurations. All memories (slave cores) can be accessed simultaneously by each master core continuously generating memory requests. Furthermore, the size of each queue (and FIFO) in the network is set to 8×32 bits and the size of the reorder buffer is set to 48 words. If the maximum burst size is set to 8, the baseline architecture utilizing a statically partitioned reorder buffer [10] can support at most six outstanding read requests in a 48-word reorder buffer (regardless of the exact size of the requests), while the proposed approach is able to embed as many requests as can be reserved in the reorder buffer, i.e., at most 48 and at least six outstanding read requests.

B. Performance Evaluation

To evaluate the performance of the proposed schemes, uniform and non-uniform/localized synthetic traffic patterns are considered separately for both configurations (A and B). These workloads provide insight into the strengths and weaknesses of the different buffer management mechanisms in the interconnection networks, and we expect applications to stand between these two synthetic traffic patterns [39]–[41]. The random traffic represents the most generic case, where each processor sends in-order read/write requests to memories with a uniform probability. Hence, the target memory and request type (read or write) are selected randomly. Eight burst sizes, from 1 to 8, are stochastically chosen according to the data length of the request. In the non-uniform mode, 70% of the traffic is local requests, where the destination memory is one hop away from the master core, and the remaining 30% of the traffic is uniformly distributed to the non-local memory modules. We also consider the hotspot traffic pattern where four memory nodes are chosen as hotspots receiving an extra portion of the traffic (10%) in addition to the regular uniform traffic [22], [23]. For the uniform and hotspot traffic profiles, we obtained very similar performance gains in each configuration, though they are not presented due to the lack of space.

Fig. 17(a) and (b) shows the simulation results under the uniform and non-uniform traffic models, respectively. In each configuration, the on-chip network utilizing the proposed NI, denoted by MS (master/slave NI) and H (hybrid NI), is compared with the network equipped with the baseline NI.

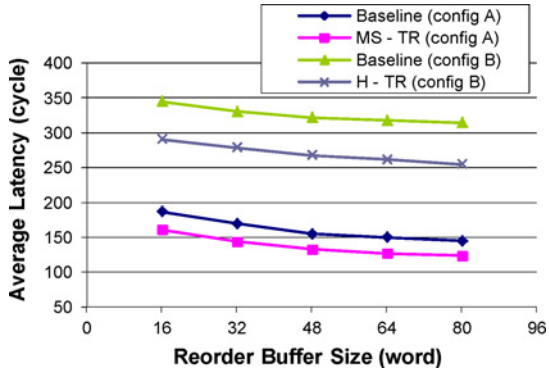


Fig. 18. Effect of reorder buffer size on the performance under the uniform traffic model.

As demonstrated in both figures, compared with the baseline architecture, the NoC using the proposed NI reduces the average latency when the request rate increases under the uniform and non-uniform traffic models. The foremost reason for such an improvement is due to employing the shared reorder buffer in the NI which allows more messages to enter the network, i.e., this leads more requests to be released from the injection queue.

We also vary the reorder buffer size to show how relative reorder buffer size affects the performance. Fig. 18 illustrates the average network latency of both configurations near the saturation point (0.6) under the uniform traffic profile. It reveals that as the reorder buffer size increases, the average network latency reduces. Given the same reorder buffer size, the proposed NI achieves better performance gain, e.g., when the reorder buffer size is 48, the performance gain for the configuration A and B is up to 16% and 21%, respectively. The proposed scheme not only achieves significant performance gains but also enables reducing the area overhead of reorder buffer by more than 60%. For instance, the proposed scheme in the configuration A with a reorder buffer size of 32 offers a better performance than a reorder buffer size of 80 in the baseline method.

Regarding the configuration B, using the master-side and slave-side NIs instead of the hybrid NI when each node is composed of a dedicated processor and memory gives a better performance, but as discussed in the next section it is not a cost-efficient approach. The hybrid structure deteriorates the performance because the buffer resources are shared. However,

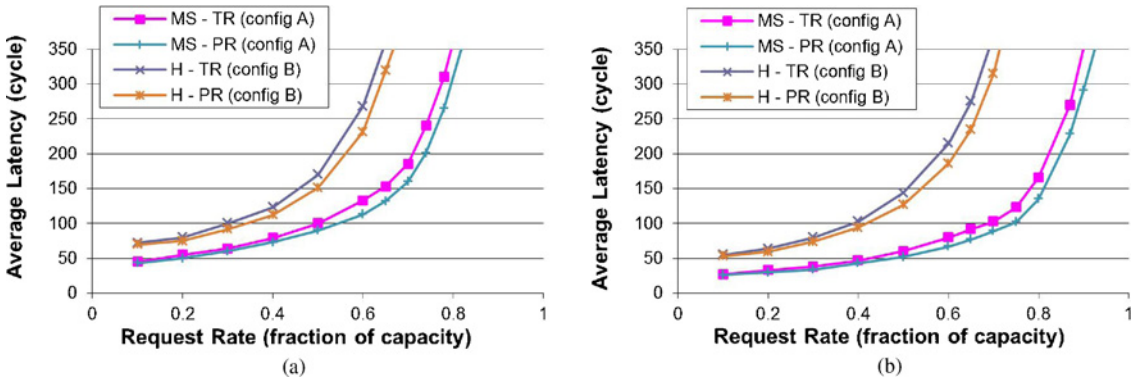


Fig. 19. Performance impact of using the PR under the (a) uniform and (b) non-uniform traffic models.

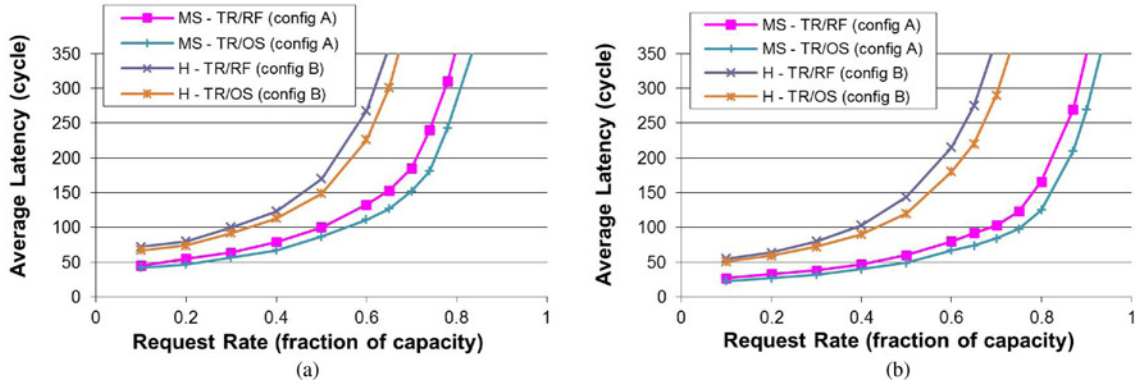


Fig. 20. Performance impact of using the OS memory controller under the (a) uniform and (b) non-uniform traffic models.

the performance given by the hybrid structure close to the saturation point is around 34% and 18% less than the other structure under the uniform and non-uniform traffic models, respectively. The performance penalty under the non-uniform traffic model is not significant compared to the uniform traffic model because the hybrid NI allows the local requests to access the local memory directly. Fig. 19(a) and (b) depicts the performance gain of the presented PR architecture under uniform and non-uniform traffic models, respectively. In each configuration the network formed by the PRs reduces the average latency compared to the network formed by TRs. The performance gain near the saturation point (0.6) under the uniform traffic model for configuration A and B is about 15% and 13%, respectively, while the hardware overhead of this router is less than 2% in comparison with the TR. This reveals that giving priority to packets according to their SN and remaining distance helps to deliver the packets to the master NI in-order. Therefore, the corresponding reserved buffer space in the reorder buffer can be released sooner and master cores can receive responses earlier than using the TRs.

To explore the impact of the proposed OS memory scheduler, we compare the network equipped with the OS scheduler with the one using the default scheduler (RF) where both networks are formed by the TRs. Fig. 20(a) and (b) presents the performance comparison between the two networks for each configuration under the uniform and non-uniform traffic models. Compared with the RF scheme, the network utilizing the OS scheduler gives significant improvements in average network latency. The performance gain of the OS scheduler close to the saturation rate under the uniform traffic model for configuration A and B is up to 17% and 16%, respectively. The average memory utilization and average memory latency are also computed near the saturation rate under the uniform traffic profile for the configuration A. According to our observation, at least half of the request buffers of each memory controller are occupied under the uniform traffic model with the given injection rate, which keeps the memory controller busy all the time. As a result, compared with the RF scheme, the average utilization of memories is improved by 22% while the average memory latency is reduced by 19%. Compared with RF, the hardware overhead of the OS scheme is negligible since both of them have similar request and data queues, buffer management, and bank interleaving arbiter. In addition, near the saturation rate under the uniform traffic profile for the configuration A the average and maximum buffer occupancy of reorder buffers is around 70% and above 90%, respectively. The average and maximum outstanding messages in the system is around 160 and 230, respectively.

C. Hardware Overhead

For appraising the area overhead of the proposed architectures, each scheme is synthesized by Synopsys D.C. using the UMC 90 nm technology with an operating point of 1 GHz and supply voltage 1 V. We perform place-and-route, using Cadence Encounter, to have precise power and area estimations. The power dissipation of each scheme, including both dynamic and leakage power, is also calculated near the saturation point (0.6) under the uniform traffic model using Synopsys Prime-

TABLE I

HARDWARE IMPLEMENTATION DETAILS

Components	Area (mm^2)	Power (mW)
Slave-side	0.0428	17
Master-side	0.0755	26
Hybrid	0.1014	37
Memory controller (OS)	0.0807	31
Memory controller (RF)	0.0798	27
TR	0.1853	65
PR	0.1881	68

Power. In addition to the aforementioned configuration of the NI, the T-ID and SN are set to 4-bit and 3-bit, respectively. The layout areas and power consumptions of the master-side, slave-side, hybrid interfaces, different memory controller, and routers are listed in Table I. As can be seen from the table, using the hybrid architecture for the latter configuration (B) is more beneficial (in terms of power and area) than using the master-side and slave-side models when each node is composed of a dedicated processor and memory. That is, using a hybrid NI model reduces 14.3% and 13.8% in hardware area and power dissipation, respectively. Because all queues (and FIFOs) are equal in size, they do not affect the comparison. On the other hand, the master-side and slave-side NI architectures are more cost efficient if each node consists of a dedicated processor or memory as in the former configuration (A). Also, comparing the area cost of the baseline model to each proposed NI indicates that the hardware overheads of implementing the proposed NI schemes are less than 0.5%. Furthermore, for the slave-side interface within the memory controller, since each of the memories utilized in this paper has four banks, four bank queues have been implemented in the memory controller.

VIII. CONCLUSION

Accessing several memory modules in parallel to augment the memory bandwidth, may lead to deadlock caused by the in-order requirement [11]. The deadlock can be solved if a reordering mechanism is exploited by the NI. The resource utilization of the conventional reordering methods is not efficient enough. Therefore, in this paper, we presented a high performance NI with a novel dynamic buffer allocation and a PR model to improve the resource utilization, and overall on-chip network performance. In addition to the resource utilization of the NI and on-chip network, the utilization of memories considerably affects the network latency. Therefore, we have developed an optimized scheduling method for the DRAM memories and integrated it in the NI such that the network and memory latencies were reduced significantly in comparison with the baseline architecture. The micro-architectures of the proposed NIs which are compatible with the AMBA AXI protocol have been presented. A cycle-accurate simulator was used to evaluate the efficiency of the proposed architecture. Under both uniform and non-uniform traffic models, in high traffic load, the proposed NI architecture has lower average delay in comparison with the baseline architecture.

REFERENCES

[1] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep.–Oct. 2007.

[2] A. Jantsch and H. Tenhunen, *Networks on Chip*. Norwell, MA: Kluwer, 2003.

[3] B. Towles and W. Dally, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, 2001, pp. 684–689.

[4] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[5] P. Lotfi-Kamran, M. Daneshtalab, C. Lucas, and Z. Navabi, "BARP-a dynamic routing protocol for balanced distribution of traffic in NoCs to avoid congestion," in *Proc. 11th ACM/IEEE DATE*, Mar. 2008, pp. 1408–1413.

[6] M. Daneshtalab, M. Ebrahimi, P. Liljeberg, J. Plosila, and H. Tenhunen, "A low-latency and memory-efficient on-chip network," in *Proc. 4th ACM/IEEE Int. Symp. NOCS*, May 2010, pp. 99–106.

[7] *AMBA AXI Protocol Specification*, ARM, Cambridge, U.K., Mar. 2004.

[8] *Open Core Protocol Specification. 2.0 Release Candidate*, OCP International Partnership, Redwood, CA, 2003.

[9] *Device Transaction Level (DTL) Protocol Specification. Version 2.2*, Philips Semiconductors, Eindhoven, The Netherlands, Jul. 2002.

[10] X. Yang, Z. Qing-Li, F. Fang-Fa, Y. Ming-Yan, and L. Cheng, "NISAR: An AXI compliant on-chip NI architecture offering transaction reordering processing," in *Proc. ASICON*, 2007, pp. 890–893.

[11] W. Kwon, S. Yoo, S. Hong, B. Min, K. Choi, and S. Eo, "A practical approach of memory access parallelization to exploit multiple off-chip DDR memories," in *Proc. DAC*, Jun. 2008, pp. 447–452.

[12] A. Radulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Trans. Comput.-Aided Des.*, vol. 24, no. 1, pp. 4–17, Jan. 2005.

[13] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proc. ISCA*, 2000, pp. 128–138.

[14] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A scalable, single-chip communication architectures," in *Proc. IEEE Int. Conf. PACT*, Oct. 2000, pp. 37–46.

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

[16] M. Lai, Z. Wang, L. Gao, H. Lu, and K. Dai, "A dynamically-allocated virtual channel architecture with congestion awareness for on-chip routers," in *Proc. 46th DAC*, 2008, pp. 630–633.

[17] G. L. Frazier and Y. Tamir, "The design and implementation of a multi-queue buffer for VLSI communication switches," in *Proc. ICCD*, Oct. 1989, pp. 466–471.

[18] 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*, 2009, pp. 1058–1063.

[19] T. Ono and M. Greenstreet, "A modular synchronizing FIFO for NoCs," in *Proc. 3rd ACM/IEEE Int. Symp. NoC*, May 2009, pp. 224–233.

[20] C. E. Cummings, "Simulation and synthesis techniques for asynchronous FIFO design," in *Proc. SNUG*, 2002, pp. 1–23.

[21] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, "Designing message-dependent deadlock free networks on chips for application-specific systems on chips," in *Proc. VLSI-SoC*, 2006, pp. 158–163.

[22] G. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel Distribut. Syst.*, vol. 11, no. 7, pp. 729–738, Jul. 2000.

[23] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. San Mateo, CA: Morgan Kaufmann, 2002.

[24] S. E. Lee, J. H. Bahn, Y. S. Yang, and N. Bagherzadeh, "A generic network interface architecture for a networked processor array (NePA)," in *Proc. ARCS*, 2008, pp. 247–260.

[25] W. Jang and D. Z. Pan, "An SDRAM-aware router for networks-on-chip," in *Proc. DAC*, 2009, pp. 800–805.

[26] Z. Fang, X. H. Sun, Y. Chen, and S. Byna, "Core-aware memory access scheduling schemes," in *Proc. IEEE IPDPS*, May 2009, pp. 1–12.

[27] *Micron 512Mb: x4, x8, x16 DDR2 SDRAM Datasheet*, Micron Technology, Inc., Boise, ID, 2006.

[28] H. G. Rotithor, R. B. Osborne, and N. Aboulenein, "Method and apparatus for out of order memory scheduling," Intel Corporation, Santa Clara, CA, U.S. Patent 7127574, Oct. 2006.

[29] J. Hu and R. Marculescu, "DyAD-smart routing for networks-on-chip," in *Proc. DAC*, 2004, pp. 260–263.

[30] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proc. Symp. Comput. Architect.*, May 1992, pp. 278–287.
[31] D. Wu, B. M. Al-Hashimi, and M. T. Schmitz, "Improving routing efficiency for network-on-chip through contention-aware input selection," in *Proc. 11th ASP-DAC*, 2006, pp. 36–41.
[32] J. Shao and B. T. Davis, "A burst scheduling access reordering mechanism," in *Proc. 13th Int. Symp. High-Performance Comput. Architect.*, Feb. 2007, pp. 285–294.
[33] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. San Mateo, CA: Morgan Kaufmann, 1998.
[34] I. Hur and C. Lin, "Memory scheduling for modern microprocessors," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, pp. 1–36, Dec. 2007.
[35] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *Proc. ISCA*, 2008, pp. 453–464.
[36] T. Kgil, A. G. Saidi, N. L. Binkert, S. K. Reinhardt, K. Flautner, and T. N. Mudge, "PicoServer: Using 3D stacking technology to build energy efficient servers," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 4, no. 4, pp. 1–34, Oct. 2008.
[37] G. L. Loi, B. Agarwal, N. Srivastava, S.-C. Lin, and T. Sherwood, "A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy," in *Proc. 43rd Des. Automat. Conf.*, 2006, pp. 991–996.
[38] *Gaisler IP Cores*, Aeroflex Gaisler, Goteborg, Sweden, 2009 [Online]. Available: http://www.gaisler.com/products/grlib
[39] R. Das, S. Eachempati, A. K. Mishra, V. Narayanan, and C. R. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," in *Proc. 15th HPCA*, 2009, pp. 175–186.
[40] M. Daneshtalab, M. Ebrahimi, T. C. Xu, P. Liljeberg, and H. Tenhunen, "A generic adaptive path-based routing method for MPSoCs," *J. Syst. Architect.*, vol. 57, no. 1, pp. 109–120, 2011.
[41] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network on chip interconnect architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

Masoud Daneshtalab (S'03) received the Bachelors degree in computer engineering from the Shahid-Bahonar University of Kerman, Kerman, Iran, in 2002, and the Masters degree in computer architecture from the School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran, in 2006. Since May 2009, he has been a doctoral candidate with the Graduate School in Electronics, Telecommunications, and Automation, Helsinki University of Technology, Helsinki, Finland. He is expected to receive the Ph.D. degree in 2011.

Since 2008, he has been with the Embedded Computer Systems Laboratory, University of Turku, Turku, Finland. He has published in more than 50 refereed international journals and conference papers. His current research interests include on/off-chip interconnection networks for multiprocessor architectures, networks-on-chip, dynamic task allocation, 3-D systems, and low-power digital design. His Ph.D. thesis is focused on adaptive implementation of on-chip networks.

Mr. Daneshtalab has served as a Program Committee Member in several different conferences, including PDP, ICESS, and DATICS.

Masoumeh Ebrahimi (S'09) received the Bachelors degree in computer engineering from the School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran, in 2005, and the Masters degree in computer architecture from Azad University, Science, and Research Branch, Tehran, in 2009.

Since 2009, she has been with the Embedded Computer Systems Laboratory, University of Turku, Turku, Finland. Her current research interests include interconnection networks, networks-on-chip (NoCs), 3-D integrated systems, and systems-on-chip. Her Ph.D. thesis is focused on routing protocols in 2-D and 3-D NoCs. She has published in more than 20 international refereed journals and conference papers.

Pasi Liljeberg (M'09) received the M.S. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively.

Currently, he is an Adjunct Professor in embedded computing architectures with the Department of Information Technology, University of Turku. Since January 2010, he has been with the Embedded Computer Systems Laboratory, University of Turku. From 2007 to 2009, he was with the Academy of Finland holding a post-doctoral position. He has established and is leading a research group focusing on reliable and fault-tolerant self-timed communication platforms for chip multiprocessor systems. His current research interests include intelligent network-on-chip (NoC) communication architectures, fault-tolerant and thermal-aware design aspects, 3-D multiprocessor system architectures, embedded computing platforms for nanoscale NoC, and reconfigurable system design.

Juha Plosila (M'06) received the M.S. and Ph.D. degrees in electronics and communication technology from the University of Turku, Turku, Finland, in 1993 and 1999, respectively.

He is an Adjunct Professor in digital systems design with the Department of Information Technology, University of Turku. For five years, from 2006 to 2011, he has been an Academy Research Fellow with the Academy of Finland. He directs an active research group focusing on modeling, design, and verification of network-on-chip (NoC) and 3-D integrated systems at different abstraction levels at the University of Turku. His current research interests include formal methods for NoC and embedded system design, reliable on-chip communication techniques, on-chip monitoring and control methods, dynamically reconfigurable systems, and application mapping on NoC-based systems.

Dr. Plosila is an Associate Editor of the *International Journal of Embedded and Real-Time Communication Systems*, published by IGI Global.

Hannu Tenhunen (M'90) received the Ph.D. degree from Cornell University, Ithaca, NY, in 1985.

Since 1985, he has been a Professor, Invited Professor, or Honorary Professor with the University of Tampere, Tampere, Finland, Stockholm University, Stockholm, Sweden, Cornell University, University of Grenoble, Grenoble, France, Shanghai University, Shanghai, China, Beijing University, Beijing, China, and the University of Hong Kong, Pokfulam, Hong Kong. In recent years, he has been an Invited Professor with the University of Turku where he has established the Embedded Computer Systems Laboratory, the leading computer architecture and systems research center in Finland. His current research interests include new computational architectures, dependability issues, on-chip and off-chip communication, and mixed signal and interference issues in complex electronic systems including 3-D integration. He has (co-)authored over 600 publications and invited key note talks internationally.