A High-Performance Network Interface Architecture for NoCs Using Reorder Buffer Sharing

Masoumeh Ebrahimi, Masoud Daneshtalab, Pasi Liljeberg, Juha Plosila, Hannu Tenhunen
Department of Information Technology, University of Turku, Turku, Finland
{masebr, masdan, paslil, juhplo, hanten}@utu.f

Abstract- Increasing memory parallelism in MPSoCs to provide higher memory bandwidth is achieved by accessing multiple memories simultaneously. Inasmuch as the response transactions of concurrent memory accesses must be in-order, a reordering mechanism is required. To our knowledge the resource utilization of conventional reordering mechanisms is low. In this paper, we present a novel network interface architecture for on-chip networks to increase the resource utilization and to improve overall performance. Also, based on the proposed architecture, a hybrid network interface is presented to integrate both memory and processor in a tile. The proposed architecture exploits AXI transaction based protocol to be compatible with existing IP cores. Experimental results with synthetic test cases demonstrate that the proposed architecture outperforms the conventional architecture in terms of latency. Also, the cost of the presented architecture is evaluated with UMC 0.09µm technology.

I. INTRODUCTION

Network-on-chip (NoC) is a feasible alternative for the traditional bus-based communication in SoCs due to its reusability, scalability, and parallelism in communication infrastructure [1][2]. NoCs are composed of routers connecting Processing Elements (PE), to deliver the data (packets) from one place to another [3], and Network Interfaces (NI) the communication interface between each PE and router. The fundamental function of network interfaces is to provide communication between PEs and network infrastructure. That is, the network interface translates the language between the PE and router based on a standard communication protocol such as AXI [4] and OCP [5]. In-order delivery should be handled when exploiting an adaptive routing algorithm for distributing the packet through the network [6], or in obtaining memory access parallelization by sending requests from a master IP core to multiple slave memories [7][8]. The former is dependent on the routing protocols of the network, whilst the latter is dominated by the distributed shared memory architecture for on-chip multiprocessor which demands higher memory bandwidth. The subtle point is that in distributed shared memory systems, the responses might need to be completed in-order even if the on-chip network exploits a deterministic routing algorithm. That is, 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 network interface. For implementing an efficient network interface, 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 network interfaces is significantly low. Therefore, the traditional buffer management is not efficient enough for network interfaces. Hence, an advantageous reordering mechanism and resourceful management of buffers in the network interface are demanded to increase the utilization and efficiency of the on-chip interconnection network.

In this work, we introduce a high performance network interface architecture utilizing a novel dynamic buffer allocation for improving the resource utilization and performance of the network interface and on-chip network. The proposed network interface architecture enables sharing slots of reorder buffer slots via a dynamic buffer management [9][10], thereby replacing the conventional, static resource allocation. Also, based on the proposed architecture, a hybrid network interface is presented to integrate both memory and processor in a tile. Besides, the proposed architecture exploits AMBA AXI protocol, to allow backward compatibility with existing IP cores [4]. We also present micro-architectures of the network interface, particularly the reordering mechanism to realize the idea. To our knowledge there are not any documented implementation details of the reordering buffer yet. The paper is organized as follows. In Section II, a brief review of the related works is presented. In Section III, the proposed architecture is discussed while the results are presented in Section IV, and the summary and conclusion are given in the last section.

II. RELATED WORK

Due to the fact that most of the recent published researches have focused on the design and description of NoC architectures, there has been relatively little attention to network interface designs particularly when supporting out-of-order mechanism. The authors in [7] present ideas of transaction ID renaming and distributed soft arbitration in the context of distributed shared memories. In such a system, because of using a 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 has been improved in [11] by moving reorder buffer resources from the network interface into network routers. In spite of increasing the resource utilization, the delay of release packets recalling data from distributed reordering buffer can significantly degrade the performance when the size of the network increases [11]. Moreover, the proposed architecture is restricted to deterministic routing algorithms and, thus, it is not a suitable method for an adaptive routing. However, neither [7] nor [11] has presented a micro-architecture of the network interface. An efficient on-chip network interface supporting shared memory abstraction and flexible network configuration is presented by Radulescu et al [8]. 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 [6], a network interface architecture using AXI protocol capable of packet reordering based on a look up table;

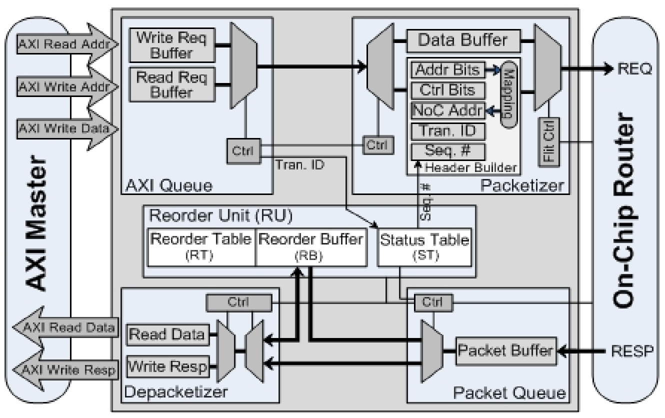

Fig. 1. Master-side network interface architecture.

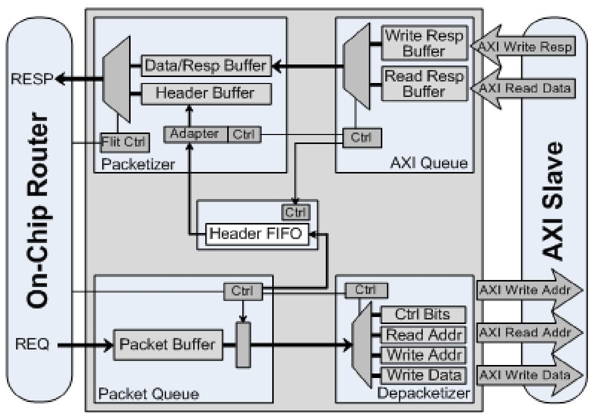

Fig. 2. Slave-side network interface architecture.

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 network interface. The major contribution of this paper is to propose a novel dynamic buffer allocation architecture for the reorder buffer to increase the utilization.

III. Proposed Network Interface Architecture

In the AXI transaction-based model [4][7], IP cores can be classified as master (active) and slave (passive) [8]. Masters 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) [8]. 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 network interface is needed to afford this ordering requirement [4][7][11]. Since IP cores are classified into masters and slaves, the network interface is also divided into the master network interface (Fig. 1) and slave network interface (Fig. 2). Both network interfaces 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 network interfaces for both master and slave sides are described in detail as follows.

A. Master-side Network Interface:

As shown in the Fig. 1, the forward path of the master network interface 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 transmits write address, write data, or read address to the network interface 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 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 assembled. Afterward, the flit controller wraps up the 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.

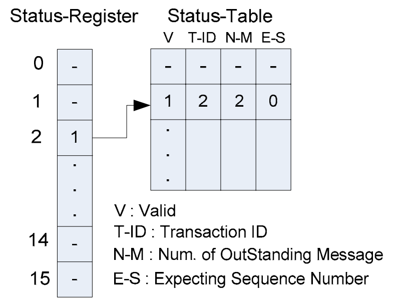

Fig. 3. Status-Register and StatusTable of Reorder Unit.

Reorder Unit: it is the most influential part of the network interface 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 (S_Reg) is an n-bit register where each bit corresponds to one of the AXI transaction IDs. As depicted in Fig. 3, 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 (S_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 Message (N-M) 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 an admittance from the reorder unit to enter the network, the corresponding bit in the status register goes high (Procedure A, line 1). The sequence number (Seq-Num) is produced by the reorder unit, if the admittance is given. This value, indicating the order of the messages within the transaction ID, is equal to zero for the first message of each transaction ID (Procedure A, line 2). "ReservedSize" keeps 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 transaction IDs. In order to prevent overflow of the reorder buffer, the reorder unit compares the new message size with the free space of reorder buffer. If the required space is available, the message will be admitted and the required space in the reorder buffer must be reserved (Procedure A, line 3). An available (free) row in the status table will be initiated by procedure B, when the second request of a transaction ID is admitted. For the rest of the admitted requests of the transaction ID, the procedure C should be executed as the sequence number is obtained by adding N-M and E-S values. Also, the number of outstanding message (N-M) is increased by +1, and the required space in the reorder buffer must is reserved by procedure C. Note that E-S indicates the next response sequence number of the corresponding transaction ID that should be delivered to the depacketizer unit.

Procedure A:
```
1  S_Reg(T_ID)                <= '1';
2  SeqNum                     <= (others =>'0');
3  ReservedSize               <= ReservedSize + NewMsgSize;
```
Procedure B:
```
1  S_Table(FreeRow)(v)        <= '1';
2  S_Table(FreeRow)(T_ID)     <= Tran_ID;
3  S_Table(FreeRow)(N_M)      <= "0010";
4  S_Table(FreeRow)(E_S)      <= (others =>'0');
5  SeqNum                     <= "001";
6  ReservedSize               <= ReservedSize + NewMsgSize;
```
Procedure C:
```
1  SeqNum                     <= S_Table(FindRow)(N_M) +
                                   S_Table(FindRow)(E_S);
2  S_Table(FindRow)(N_M)      <= S_Table(FindRow)(N_M) + 1;
3  ReservedSize               <= ReservedSize + NewMsgSize;
```

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 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 which releases the occupied buffer space; thereafter, E-S and N-T values will be increased by +1 and -1, respectively (Procedure D).

If N-M value reaches zero, the transaction will be terminated by resetting the valid bit for both status register and status table. However, the packet is out-of-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.

Procedure D:
```
1  S_Table(FindRow)(N_M)   <= S_Table(FindRow)(N_M) - 1;
2  S_Table(FindRow)(E_S)   <= S_Table(FindRow)(E_S) + 1;
3  ReservedSize            <= ReservedSize - ReceivedMsgSize;
```

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.

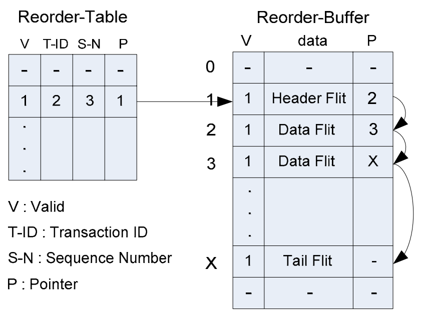

Fig. 4. Dynamic buffer allocation

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. 4 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 according to the procedure E, and F. The first three operations in the procedure E, stores the transaction ID and sequence number 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 E updates the pointer to point to the available slot in the reorder buffer.

Procedure E:
```
ReorderTable [FreeRow][V]        <= '1';
ReorderTable [FreeRow][T-ID]     <= HeaderFlit[TranID];
ReorderTable [FreeRow][S-N]      <= HeaderFlit[SeqNum];
ReorderTable [FreeRow][P]        <= Current_Free_Slot;
```
Procedure F:
```
ReorderBuf[Current_Free_Slot][V]     <= '1';
ReorderBuf[Current_Free_Slot][Data]  <= flit;
ReorderBuf[Current_Free_Slot][P]     <= Next_Free_Slot;
Current_Free_Slot                    <= Next_Free_Slot;
```

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 in order to store the current flit, Next_Free_Slot returns an available slot for the next flit. By repeating the operations in the procedure F, whole of the payload flits will be stored in the reorder buffer.

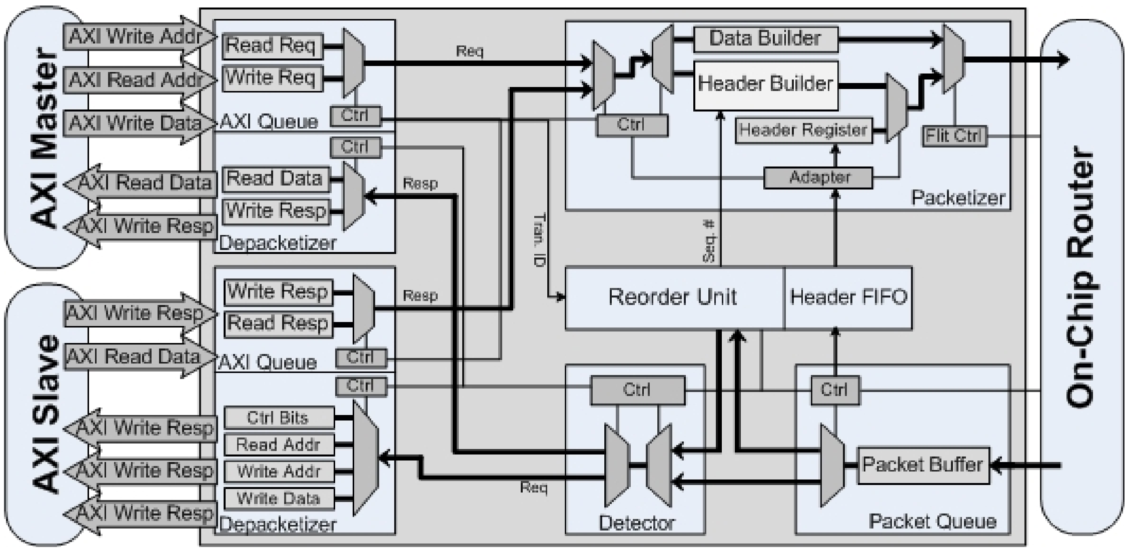

Fig. 5. Hybrid network interface architecture.

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 Network Interface:

A slave IP core cannot operate independently. It receives requests from master cores and responds to them. Hence, using reordering mechanism in the slave network interface 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.

C. Hybrid Network Interface

The hybrid model is formed by combining the *master-side* and *slave-side* network interfaces*.* As illustrated in Fig. 5, 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 network interfaces (master and slave), the hybrid model is more beneficial, particularly in terms of area and power costs.

IV. EXPERIMENTAL RESULTS

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 network interface, 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 network interface) and other fifteen nodes are memories (slave cores-with slave network interface). For the second configuration (B), each node is considered to have a processor and a memory (master and slave cores-with hybrid network interface). 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. Packets of different message types (read and write) are assigned to corresponding VCs to avoid message deadlock [12]. The arbitration scheme of the switch allocator is round-robin. 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 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 the master and the time when the response is completely delivered to the master from the memory. The request rate is defined as the ratio of the successful read/write request injections into the network interface over the total number of injection attempts. All the cores and routers are assumed to operate at 2GHz. For fair comparison, we keep the bisection bandwidth constant in all configurations. We also set the size of the reorder buffer to 48 words, able to embed 6 outstanding requests with the burst size of 8. All memories (slave cores) can be accessed simultaneously by each master core continuously generating memory requests.

A. Performance evaluation

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. 6 reveals that compared with the baseline architecture [6][7] the proposed architecture reduces the average latency when the request rate increases in both configurations 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 allows more messages to enter the network.

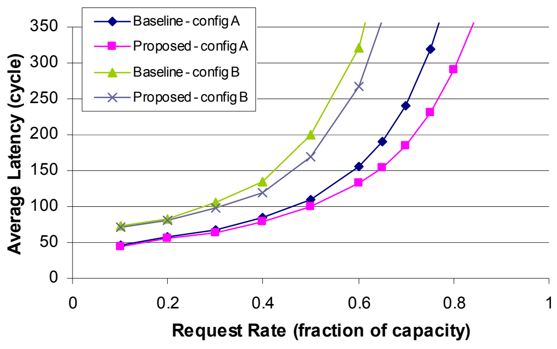

Fig. 6. Performance evaluation of both configurations.

B. *Hardware Overhead*

For appraising the area overhead of the proposed architectures, the network interfaces were synthesized by Synopsys D.C. using the UMC 0.09µm technology. In addition to the aforementioned configuration of the network interface, the tran_id and seq_id were set to 4-bit and 3-bit respectively. The layout areas and power consumptions of the master-side, slave-side, and hybrid interfaces are listed in Table 1. As can be seen from the table, using the hybrid architecture with the later configuration (B) is more beneficial 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 network interface model reduces 14.3% and 13.7% in hardware area and power dissipation respectively. On the other hand, the master-side and slave-side network interfaces 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 for each proposed network interface indicates that the hardware overheads of implementing the proposed schemes are less than 0.5%.

V. Summary and Conclusion

Accessing several memories in parallel to augment the memory bandwidth, may lead to the deadlock caused by the in-order requirement [7]. The deadlock can be solved if a reordering mechanism is exploited by the network interface. The resource utilization of the conventional reordering methods is not efficient enough; thus, in this work, we presented a high performance network interface with a novel dynamic buffer allocation which improves the resource utilization, and overall on-chip network performance. Also, the micro-architectures of the proposed master-side and slave-side network interfaces 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 both uniform and non-uniform traffic models, in high traffic load, the proposed architecture had lower average communication delay in comparison with the baseline architecture.

Table 1. Hardware implementation details.

NI	Area (μm^2)	Power (μW)
Slave-side	42848	2.74
Master-side	75559	4.41
Hybrid	101492	6.12

Acknowledgment

The authors wish to acknowledge Nokia Foundation for the partial financial support during the course of this research.

References

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

[2] L.Benini and G.De Micheli, “Networks on chips: a new SoC paradigm”, IEEE Computer, January 2002.

[3] 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.

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

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

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] S. Murali, and et al. “Designing message-dependent deadlock free networks on chips for application-specific systems on chips,” In Proc. VLSI-SoC, pages 158-163, 2006.