Connection-oriented Multicasting in Wormhole-switched Networks on Chip

Zhonghai Lu, Bei Yin and Axel Jantsch
Laboratory of Electronics and Computer Systems
Royal Institute of Technology, Sweden
{zhonghai,axel}@imit.kth.se, {beiy}@kth.se

Abstract

Network-on-Chip (NoC) proposes networks to replace buses as a scalable global communication interconnect for future SoC designs. However, a bus is very efficient in broadcasting. As the system size scales up to explore the chip capacity, broadcasting in NoCs must be efficiently supported. This paper presents a novel multicast scheme in wormhole-switched NoCs. By this scheme, a multicast procedure consists of establishment, communication and release phase. A multicast group can request to reserve virtual channels during establishment and has priority on arbitration of link bandwidth. This multicasting method has been effectively implemented in a mesh network with deadlock freedom. Our experiments show that the multicast technique improves throughput, and does not exhibit significant impact on unicast performance in a network with mixed unicast and multicast traffic if the network is not saturated.

1 Introduction

As the technology steadily scales, chip design is increasingly becoming communication-bound. Network-on-Chip [1, 5, 10] addresses the design challenges by proposing networks to replace buses as a scalable global communication platform. In a NoC, heterogeneous resources such as processors, DSPs, FPGAs/ASICs, and memories are interconnected by switches. These resources communicate by routing packets instead of using dedicated wires.

Buses (a single bus, segmented or crossbar-type buses and a hierarchy of buses) do not scale well with the system size in bandwidth and clocking frequency. However, a bus is very efficient in broadcasting since all clients are directly connected to it. A network allows many more concurrent transactions, but it does not directly support multicast. As there exists a variety of SoC applications, many applications necessitates to support multicast in the case of passing global states, managing and configuring the network, and implementing cache coherency protocols etc. Particularly, real-time constrained, throughput-oriented embedded applications for multi-media processing will demand an efficient means to implement multicast. One crucial aspect for supporting multicast in SoCs is Quality-of-Service (QoS), which means that the performance of multicast traffic should be predictable. Implementing multicast by sending multiple unicast messages is neither efficient nor scalable. In addition, in a network with a mixture of unicast and multicast traffic, multicast traffic should not degrade the performance of unicast traffic since multicast traffic takes only a portion of the total network traffic.

In this paper we present a connection-oriented multicast scheme in wormhole-switched networks on chip. Wormhole switching [3] is a network flow control mechanism that allocates buffers and physical channels (PCs) to flits instead of packets. A packet is encapsulated into one or more flits. A flit, the smallest unit on which flow control is performed, can advance once buffering in the next hop is available to hold the flit. This results in that the flits of a packet are delivered in a pipeline fashion. In order to make an efficient use of link bandwidth, wormhole switching can employ *virtual channels* (VCs or lanes) to enhance throughput [2]. Because of these advantages, namely, *better performance*, *smaller buffering requirement* and *greater throughput*, wormhole switching with lanes is being advocated for on-chip networks [5, 8].

By our multicast scheme, multicasting consists of three phases: *group setup*, *communication*, and *group release*. A multicast is realized by sending a single copy of multicast packets to multicast group members along a pre-established path. This results in low packet overhead for multicasting. During the setup phase, multicasting can be aware of QoS in the sense that a multicast group may request to reserve VCs, and enjoy a higher priority against unicast packets for link bandwidth arbitration. Although the three-phase (setup, transmission, and release) communication has been used for establishing virtual-circuit communication to support QoS in store-and-forward packet-switched networks, applying the technique to implement multicast in a wormhole-switched network on chip is the novel aspect of this paper. Moreover, we shall look at how much impact multicast traffic will exert on unicast traffic, and the performance tradeoff between *multicast without VC reservation* and *multicast with VC reservation*.

2 Related Work

Multicasting in wormhole-switched networks has been extensively studied in parallel machines in order to support collective communications such as barrier synchronization, reduction and global combining [6, 9]. Multicast can be achieved via software or hardware approach. With software implementation of multicast, a multicast operation is implemented by sending a separate copy of the messages from the source node to every destination or to a subset of destinations, each of which in turn forwards the message to one or more other destinations in a multicast tree.

As the software approach is not efficient enough, hardware support of multicast communication is proposed. With the *tree-based multicast* [9], the destination set is partitioned at the source, and separate copies of the message are transmitted. A message may be replicated at intermediate nodes and forwarded to disjoint subsets of destinations in the tree. This scheme does not perform well to be deadlock free unless messages are very short because the entire tree is blocked if any of its branches are blocked. A solution is to forbid branching at intermediate nodes, leading to a multicast path pattern, called *path-based multicast* [6]. In order to reduce the length of the multicast path, the set of destination nodes may be divided into multiple disjoint subsets. A copy of the source message is sent across several multicast paths, each path for each subset of the destination nodes. In this scheme, multicasting is realized by sending multi-destination messages. The header of multi-destination messages must carry the addresses of all the destination nodes. As the header is an overhead, the message latency is increased and the effective network bandwidth is reduced. Besides, multicast traffic does not reserve network resources, equally competing with unicast traffic for buffers and link bandwidth, resulting in no QoS for multicasting.

By the traditional multicast schemes, group formation and multicast communication are not decoupled. The multicast-packet overhead is high and there is no QoS concern. In our multicast scheme, there is an explicit multicast group setup phase. After a group is set up, multi-destination messages carry only the group identity number not the addresses of all the destination nodes. In addition, a multicast group can be aware of QoS by reserving lanes for performance enhancement.

For a circuit-switched network on chip, a multicasting scheme using global traffic information is proposed in [7]. This scheme is difficult to scale to a large system size since it relies on the global network state. Connection-oriented communication has been proposed in the Mango [1] and Æthereal [5] NoCs to achieve QoS for unicasting. In Mango, connections are created using asynchronous/clockless circuitry. By reserving link bandwidth, Æthereal builds connections to provide a virtual contention-less path from sources to destinations. We use the connection-oriented technique to realize QoS-aware multicasting in a best-effort network. As stated, the contention-free route in the Æthereal NoC [5] and the looped containers [10] in the Nostrum NoC can be used to realize multicasting. But no concrete results are released so far.

3 The Multicast Scheme

3.1 Unicast in wormhole networks

Figure 1 sketches an input-buffering wormhole switch with lanes. It employs credit-based link-level flow control to coordinate packet delivery between switches.

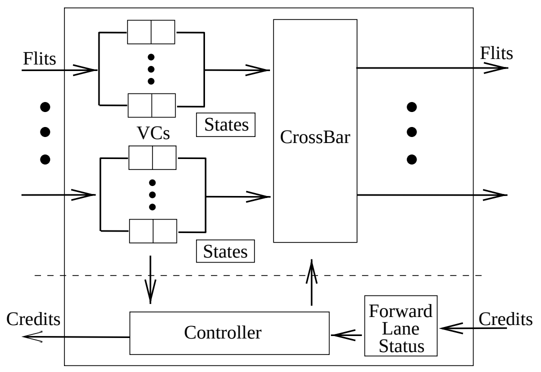

Figure 1: An input-buffering wormhole switch

A packet is segmented into flits, which are then delivered in the network. After the segmentation, a packet is typically composed of a head flit, a tail flit and body flit(s). A single-flit packet is also possible. A packet passes the switch through four states: *routing*, *lane allocation*, *flit scheduling*, and *switch arbitration*. In the routing state, the routing logic determines the routing path the packet advances. Routing is performed only when the head flit of a packet becomes the earliest-come flit in the lane. This means that if flits of a previous packet still stay in the lane, the routing will not be performed. Only when the earlier-coming flits are switched out, the head flit becomes the earliest-come flit. Then routing is performed, and the packet path and output physical channel are determined. In the state of lane allocation, the lane allocator *associates* the lane the packet occupies with an available lane in the next hop on its routing path, i.e., to make a *lane-to-lane* association. Note that it is not necessarily required that there is an empty buffer in the lane in order for the lane to be associated or allocated. A lane-to-lane association fails when all requested lanes in the next hop are already associated to other lanes in directly connected switches. If the lane-to-lane association succeeds, the packet enters into the scheduling state. If there is a buffer available in the associated lane, the lane enters into the switch arbitration. The first level of arbitration is performed on the lanes sharing the same physical channel. The second level of arbitration is for the crossbar traversal to output physical channels. If the lane wins the two levels of arbitration, the earliest-come flit in the lane is switched out. Otherwise, the lane returns back to the scheduling state. Once the tail flit is switched out, the lane-to-lane association is released, thus the allocated lane is available to be used by other packets. Credits are passed between adjacent switches in order to keep track of the status of downstream/forward lanes, such as if a lane is free, and a count of available buffers in the lane.

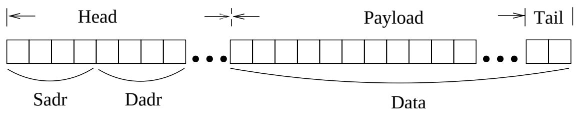

Figure 2: Unicast packet format

Figure 2 shows a typical unicast packet format, which consists of a head, a payload and a tail. The head and tail are the overhead for transmitting the payload. The head typically consists of routing and sequencing information. Basic routing information includes source address (sadr) and destination address (dadr). A switch uses the destination address to perform routing and switches the packet to the right output physical channel (PC). When the packet is split into flits, each flit contains a flit type field to identify if it is a head (H), body (B), tail (T) flit, or a single-flit packet.

3.2 The multicasting protocol

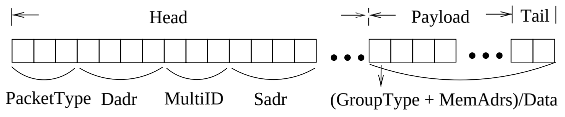

Figure 3: Multicast packet format

In order to support multicasting, we expand the packet format into that shown in Figure 3. We explain the packet fields as follows:

- **PacketType** indicates the purpose of a packet. It has six options, namely, *unicast, multicast setup, multicast setup response, multicast data, multicast group release* and *multicast group release acknowledgment.*
- **Dadr**: the destination address. In the case of multicast, it is the address of the next group member.
- **MultiID**: the multicast group identity number, which is unique for each multicast group to be established.
- **Sadr**: the source address. In the case of multicast, it is the address of the node that initiates the multicast setup. This node is called *group master*.
- **GroupType**: the type of the multicast group. It is used to inform the switches whether the multicast group will reserve a lane or not. It occupies only one bit. One group is allowed to reserve only one lane in a switch since the number of lanes is limited.
- **MemAdrs**: the multicast members' addresses. The order of the address list specifies the multicast path [1]. Specifically, the node with the first address in the list will be reached first and then second and so on. Upon reaching the node with address **Dadr**, the next member address in the list will replace the **Dadr** field.

Fields **GroupType** and **MemAdrs** are only needed for multicast setup packets, **MultiID** for multicast packets. A response packet for multicast group setup or release is handled as a unicast packet. By our scheme, a multicast group can be established and released dynamically. A multicasting procedure consists of three phases explained as follows:

1. *Group establishment*: First the group master sends a setup packet, which passes downstream to all the group member nodes along the predetermined path as indicated by **MemAdrs**. When the setup packet reaches a node, the switch records the multicast information and reserves resources according to the group type. This record will be used later to transmit multicast data. If the setup packet reaches the last group member, a setup response packet will be sent back to the group master to acknowledge the success. If the setup fails in a node, for example, due to lane unavailability, a response packet will be sent back to the master from the current node.

2. *Multicast communication*: After a successful setup, the master can send multicast data packets. The packets carrying **MultiID** will be transmitted along the same path and the same VCs which the setup packet used before. When a data packet reaches a destination node in the group, its payload is replicated and the packet is forwarded to the next member. In this way, all the members will receive the packet. The group members can also send multicast data packets, but only to the members in the downstream since a multicasting path is simplex and thus only simplex communication is allowed.

3. *Group release*: A group can only be released by its master by sending a release packet to its members. When the release packet reaches a node, the multicast record in the switch and the reserved lane will be freed after all on-going group transactions complete. Upon reaching the last member, a release acknowledgment is sent back to the master.

[1]In our approach, the multicast setup path can be diverse, and it shall follow the path-based schemes. But this is not the focus of the paper.

3.3 Multicast implementation

3.3.1 Extending the unicast switch

We have implemented the unicast switch in VHDL according to the model shown in Figure 1. The implementation consists of a data path and a control path. The data path is concerned with the flit movement through the crossbar and virtual channel. The control path realizes the functionality of the controller. For flit ejection, we implement a p-sink model to reduce cost [8]. By this model, a switch uses p flit sinks to eject flits, where p is typically equal to the number of PCs per switch. These p sinks are shared by the $p \cdot v$ lanes, where v is the number of VCs per PC.

Based on the unicast switch model, we have implemented the multicast scheme. The resultant switch supports both unicast and multicast. The data path is maintained the same as unicast while the control path is complicated. Specifically, the controller is extended to distinguish different packet types and perform actions according to the protocol. The switch must record the multicast information for each group passing it. The record of a multicast group includes $\{$**MultiID**, **GroupType**, **Sadr**, **VCID**, **VCID downstream**, **output PC**, **next member adr.**$\}$, where **VCID** is the identity number of the lane a multicast packet passes in the current switch; **VCID downstream** is the lane allocated downstream; **output PC** is the output physical channel the packet is to be switched out; **GroupType** indicates if the group reserves a lane or not. The current implementation arbitrates link bandwidth in favor of multicast traffic.

3.3.2 Deadlock avoidance

Deadlock is catastrophic to a network. It happens when a packet waits for an event that cannot happen. For example, a group of packets are unable to make progress because of waiting on one another to release buffers or channels. Forbidding such a cyclic resource dependency is a sufficient condition to design a deadlock-free network. Deadlock is related to many factors such as the network topology, flow control scheme, communication protocol and so on. Restricting the routing choice and adding buffer classes are the basic ways to deal with it.

Our multicast scheme has been implemented in a 2D mesh network employing dimension-order XY routing, which is proven to be deadlock free for unicast traffic on meshes. We constrain that a multicast path follows XY routing. This removes cyclic dependencies involving the two dimensions. However, care must be taken when planning a multicast path. Resulting from an improper path, a multicast packet may involve the turn from Y to X. As shown in Figure 4, a cycle is formed if the group is organized as $A \rightarrow B \rightarrow C \rightarrow D$. To avoid such a cycle, a group path must be organized so that no turn from Y to X is resultant. For example, the group may be organized as $A \rightarrow C \rightarrow B \rightarrow D$. This simplification avoids deadlock at the expense of restrictions on planning groups and paths.

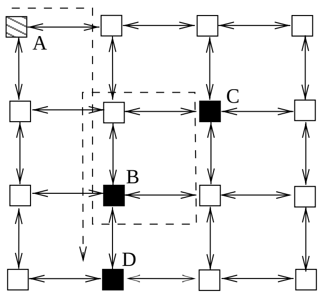

Figure 4: A cycle in a group

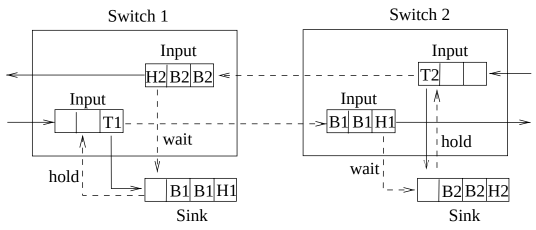

Figure 5: Deadlock while sinking and forwarding

Since a multicast packet has to sink locally at a member node and meanwhile to be forwarded downstream, the allocation of both a local sink and a lane in the next hop must be successful. In order not to introduce extra buffers and complicate the control, we decide to sink a flit when both the allocation conditions are true. This means that we perform sinking and forwarding on a multicast flit simultaneously. This may lead to deadlock with the p-sink ejection. As illustrated in Figure 5, two four-flit multicast packets pass two adjacent switches. While sinking and forwarding, both packets hold the sink the other waits for. The dashed lines in Figure 5 shows the wait-for graph [3], which forms a dependency cycle. Two solutions may be used to remove the cycle. One is to make the lane size long enough to hold an entire packet. The other is to ensure that the actual number p of sinks is larger than the number of multicast groups passing a switch, and a multicast packet does not wait for the availability of a particular sink. This guarantees that there is at least one sink available to break a possible dependency cycle due to the exhaustion of sink resources.

When a multicast setup fails, a negative response (nack) packet will be sent from the failing switch back to the group master, which in turn will send a release packet to the network. This may create a dependent loop (A positive response does not cause a loop). By adopting the technique of the credit-based end-to-end flow control and separate buffer classes in [4], we are certain that this never causes deadlock.

4 Experiments

The purposes of our experiments are to (1) compare multicasting with unicasting multiple packets; (2) investigate the impact of multicast traffic on unicast traffic in a mixed unicast-multicast network; (3) evaluate the multicast scheme with/without lane reservation.

Using the multicast-supported wormhole model, we construct a 1×7 and 4×4 mesh. The networks operate synchronously. With the switch model, it takes 5 cycles for a head flit and 3 cycles for other flits to pass through a switch. Each switch has the same configuration parameters as follows: the number of VCs per PC is four for the 1×7 mesh and six for the 4×4 mesh; the depth of a VC is two, which is the minimal number in order to pipeline flits; the number of sinks is eight for the 1×7 mesh and 24 for the 4×4 mesh. Four-flit packets are injected into the network synchronously at a constant rate. Each node has a workload of 1000 packets. Simulations terminate when anyone of the nodes completes transmission. Latency of a packet is recorded from the instant that the packet is queued in the source FIFO to that the packet is ejected from the network. Network load is the average percentage of active links through the simulation cycles. Throughput is defined as the number of packets received per cycle per node.

4.1 Multicast vs. unicast

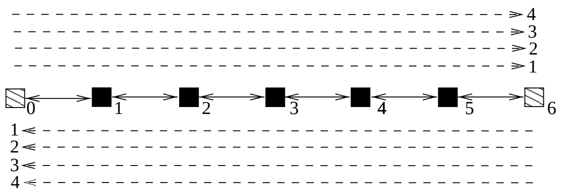

Figure 6: Traffic scenario in the 7-ary 1-mesh

As shown in Figure 6, we use the seven-node array in this set of experiments. Either unicast or multicast packets exist in the network. In the case of pure unicast, sending packets to multiple destinations is implemented by unicasting. In the experiments, node 0 and 6 send packets to the other six nodes randomly. In the multicast setting, four groups per direction are created, and the multicast groups do not reserve lanes. Node 0 and 6 are the group masters, which send multicast packets to the four groups alternatively.

Figure 7 depicts the results. With the same injection rate, the network is more loaded with the multicast, since multicast packets are delivered to all other nodes until reaching the other end while a unicast packet is sent to a particular node. The latency is worse with the multicast packets, this is due to the co-allocation of lane and sink for a multicast packet. However, the throughput of the multicast case is six

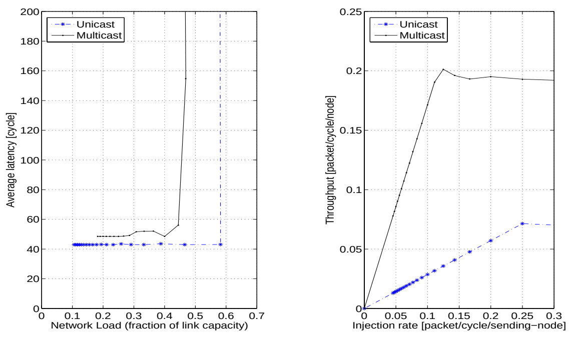

Figure 7: Multicast vs. unicast performance

times as much as that of the unicast case before the multicast network reaches saturation.

4.2 Multicast vs. mixed traffic

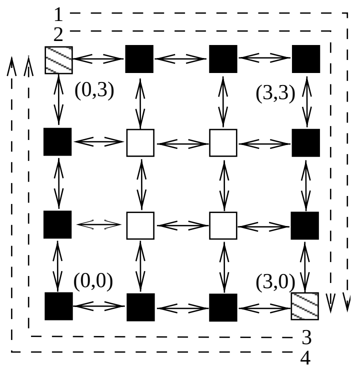

Figure 8: Traffic scenario in the mesh

The 4×4 mesh is used, and two traffic scenarios are created for the following experiments. One is *purely unicast traffic*. All network nodes send unicast packets to random destinations except themselves. The other is a *mixed unicast-multicast* scenario where four multicast groups start establishment upon simulation starts (meanwhile, unicast traffic is also injected.). As illustrated in Figure 8, four multicast groups are set up. From node (0, 3) to node (3, 0), group 1 and 2 are built following +X to -Y; from node (3, 0) to node (0, 3), group 3 and 4 are built following -X to +Y. Only the group masters send multicast packets to their members. They send one multicast packet every four packets. If a multicast packet is sent to n members, the amount of traffic is counted as n packets. The resultant multicast traffic takes 16.2% percent of the total network traffic.

We consider two cases. **Case 1**: the multicast groups do not reserve VCs (lanes); **Case 2**: the multicast groups reserve VCs. Figure 9 and Figure 10 draw the network per-

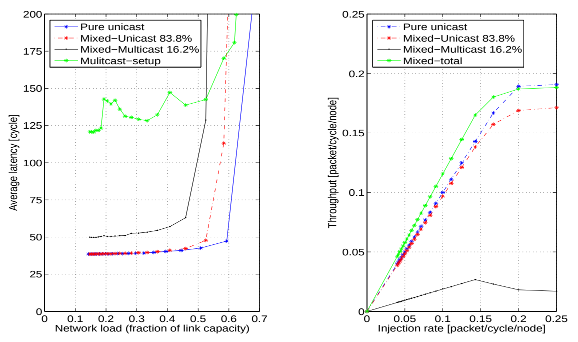

Figure 9: Performance without VC reservation

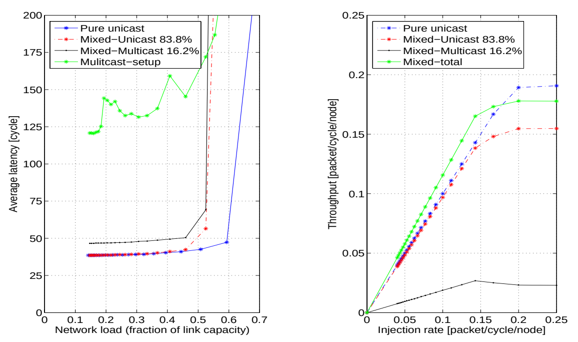

Figure 10: Performance with VC reservation

formance with the two scenarios for case 1 and case 2, respectively. In comparison with the purely unicast traffic, the unicast in the mixed traffic scenario performs equivalently to each other when the network load is below 0.45 for both cases. This means that the multicast traffic does not degrade the unicast performance if the network is not overloaded. The throughput is higher with the mixed traffic if the network is not saturated, since a multicast uses link bandwidth more efficiently. As shown in Figure 10, if a group reserves lanes, the average latency of the multicast traffic is improved 4.6 cycles on average if the network operates below load 0.45. However, due to lane reservation, the network saturation throughput is decreased by 4.2% from 0.185 to 0.177 packet/cycle/node.

As can be observed in Figure 9 and 10, the average group setup latency is 3-4 times as much as the average latency of unicast packets even when the network is not overloaded, since a group setup takes at least a round-trip time. This overhead suggests that our multicast scheme is beneficial to send block data where the amount of multicast traffic is high, if a multicast group is to be established dynamically. If a group is set up statically during the system warm-up phase, this overhead may be ameliorated.

5 Conclusions

We have presented our multicasting scheme in wormhole-switched networks on chip. With this scheme, multicasting starts after a multicast group is established. During establishment, a multicast group can reserve virtual channels. Our experimental results suggest that multicasting is beneficial in throughput. In addition, in a network with mixed unicast and multicast traffic, the multicast traffic does not show negative impact on the performance of unicast traffic if the network is not saturated. With the lane reservation, the latency of multicast traffic can be improved at the expense of slightly decreased throughput.

In future work, we aim at designing a synthesizable wormhole switch supporting the multicast scheme in order to obtain its cost overhead in area and speed penalty. The optimization of the controller will be essential for enhancing the hardware speed. Another direction is to use the QoS-aware multicast communication to emulate traditional buses. This could potentially address the problem of large amount of legacy code written for buses.

References

[1] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the Design, Automation and Test in Europe Conference*, volume 2, pages 1226–1231, 2005.

[2] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–204, March 1992.

[3] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufman Publishers, 2004.

[4] B. Gebremichael, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, and A. Rădulescu. Deadlock prevention in the Æthereal protocol. In D. Borrione and W. Paul, editors, *Proc. Working Conference on Correct Hardware Design and Verification Methods (CHARME)*, volume 3725 of *Lecture Notes in Computer Science (LNCS)*, pages 345–348, Oct. 2005.

[5] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

[6] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):793–804, 1994.

[7] J. Liu, L.-R. Zheng, and H. Tenhunen. Interconnect intellectual property for network-on-chip. *Journal of System Architectures, Special issue on networks on chip*, 50(2):65–79, February 2004.

[8] Z. Lu and A. Jantsch. Flit ejection in on-chip wormhole-switched networks with virtual channels. In *Proceedings of the IEEE Norchip Conference*, November 2004.

[9] M. P. Malumbres, J. Duato, and J. Torrellas. An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, 1996.

[10] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test in Europe Conference*, 2004.