Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the *Nostrum* Network on Chip

Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch

Laboratory of Electronic & Computer Systems, Royal Institute of Technology (KTH)
LECS/IMIT/KTH, Electrum 229, 164 40 Kista, Sweden
{micke, erlandn, thid, axel}@imit.kth.se

Abstract

In today's emerging Network-on-Chips, there is a need for different traffic classes with different Quality-of-Service guarantees. Within our NoC architecture Nostrum*, we have implemented a service of Guaranteed Bandwidth (GB), and latency, in addition to the already existing service of Best-Effort (BE) packet delivery. The guaranteed bandwidth is accessed via Virtual Circuits (VC). The VCs are implemented using a combination of two concepts that we call 'Looped Containers' and 'Temporally Disjoint Networks'. The Looped Containers are used to guarantee access to the network – independently of the current network load without dropping packets; and the TDNs are used in order to achieve several VCs, plus ordinary BE traffic, in the network. The TDNs are a consequence of the deflective routing policy used, and gives rise to an explicit time-division-multiplexing within the network. To prove our concept an HDL implementation has been synthesised and simulated. The cost in terms of additional hardware needed, as well as additional bandwidth is very low – less than 2 percent in both cases! Simulations showed that ordinary BE traffic is practically unaffected by the VCs.*

1 Introduction

Current core based *System-on-Chip (SoC)* methodologies do not offer the required amount of reuse to enable the system designer to meet the time to market constraint. A future SoC methodology should have potential of not only reusing cores but also reusing the interconnection and communication infrastructure among cores.

The need to organise a large number of cores on a chip using a standard interconnection infrastructure has been realised for quite some time. This has led to proposals for platform based designs using standardised interfaces, e.g. the VSI initiative [1]. Platforms usually contain bus based interconnection infrastructures, where a designer can create a new system by configuring and programming the cores connected to the busses. A concrete example of this is manifested in Sonic's μ-networks [2]. Due to the need for a systematic approach for designing on-chip communication Benini and Wielage [3, 4], have proposed communication centric design methodologies. They recognise the fact that interconnection and communication among cores for a SoC will captivate the major portion of the design and test effort.

As recognised by Guerrier [5], bus based platforms suffer from limited scalability and poor performance for large systems. This has led to proposals for building regular packet switched networks on chip as suggested by Dally, Sgroi, and Kumar [6, 7, 8]. These *Network-on-Chips (NoCs)* are the network based communication solution for SoCs. They allow reuse of the communication infrastructure across many products thus reducing design-and-test effort as well as time-to-market. However, if these NoCs should be useful, different traffic classes must be offered, as argued by Goossens [9]. One of the traffic classes that will be requested is the *Guaranteed Bandwidth (GB)* that has been implemented in, e.g. Philips's Æthereal [9].

Our contribution is the service of GB, to be used within our NoC architecture *Nostrum*, in addition to the already existing service of Best-Effort (BE) packet delivery [10]. *Nostrum* targets low overhead in terms of hardware and energy usage in combination with tolerance against network disturbances, e.g. congestions. In order to achieve these goals deflective routing was chosen as switching policy. In comparison to the switch of Rijpkema [11], and in Philips's Æthereal, the need for hardware is reduced by the absence of routing tables as well as in and output packet queues.

The service of GB is accessed via *Virtual Circuits (VC)*. These VCs are implemented using a combination of the two concepts called *Looped Containers* in *Temporally Disjoint Networks (TDN)*. The solution is cheap, both in terms of header information in the packets, hardware need, and bandwidth used for providing the service.

The rest of the paper is organised as follows. In section 2, we briefly describe the *Nostrum* NoC. Section 3 explains the theory behind our concept. Section 4 presents how the concept can be used and what possibilities this usage gives. The section also includes synthesis and simulation results. The last section is used for conclusions.

2 *Nostrum*

We have developed a concept that we call *Nostrum* [12] that is used for defining a concrete architecture – the *Nostrum Mesh Architecture*. The communication infrastructure used within the concept is called the *Nostrum Backbone*.

2.1. The *Nostrum* Concept

Nostrum is our concept of network based communication for 'System on Chip's (SoCs). *Nostrum* mixes traditional mapping of applications to hardware with the use of the communication infrastructure offered by Network-on-chip (NoCs). Within *Nostrum*, the '*System'* in SoC can be seen as a system of applications. An application consists of one or more processes that can be seen as functional parts of the application. In order to let these processes communicate, the *Nostrum* concept offers a packet switched communication platform and it can be reused for a large number of SoCs, since it is inherently scalable.

To make the packet switched communication practical for on-chip communication, the protocols used in traditional computer networks cannot be employed directly; the protocols need to be simplified so that the implementation cost as well as speed/throughput performance is acceptable. These simplifications are made from a functional point of view and only a limited set of functions are realised.

2.2. The *Nostrum* Backbone

The purpose of the backbone is to provide a reliable communication infrastructure, where the designer can explore and chose from a set of implementations with different levels of reliability, complexity of service etc.

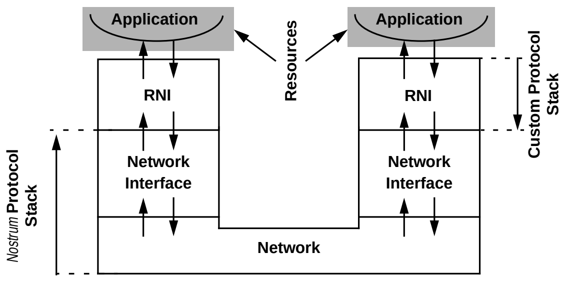

Fig. 1. The Application/RNI/NI

In order to make the resources communicate over the network, every resource is equipped with a *Network Interface (NI)*. The NI provides a standard set of services, defined within the *Nostrum* concept, which can be utilised by a *Resource Network Interface (RNI)* or by the resource directly. The role of the RNI is to act as glue (or adaptor) between the resource's internal communication infrastructure and the standard set of services of the NI. Dependent on the functionality requested from the *Nostrum Backbone*, the *Nostrum* protocol stack can be more or less shallow. However, the depth of the custom protocol stack, which may include the RNI, is not specified within the concept.

2.3. Communication Services

The backbone has been developed with a set of different communication protocols in mind e.g MPI [16]. Consequently, the backbone can be used for both BE using single-message passing between resources (datagram based communication) as well as for GB using stream oriented data distribution (VC). The message passing between the resources is packet based, i.e. the message is segmented and put into packets that are sent over the network. The ordering of packets and de-segmentation of messages is handled by the NI. In order to cover the different needs of communication two different policies are implemented:

A. Best-Effort

In the BE implementation, the packet transmission is handled by datagrams. The switching decisions are made locally in the switches on a dynamic/non-deterministic basis for every individual datagram that is routed through the network. The benefit is low set-up time for transmission and robustness against network link congestion and failure. The policy is described in [10] and will not further be discussed.

B. Guaranteed Bandwidth

The GB is the main topic of the paper and is implemented by using a packet type, which we call container. A container packet differs from the datagram packets in two ways. They follow a pre-defined route and they can be flagged as empty.

2.4. The *Nostrum* Mesh Architecture

The NoC *Nostrum* Mesh Architecture [13] is a concrete instance of the *Nostrum* concept and consists of *Resources (R)* and *Switches (S)* organised, logically and physically in

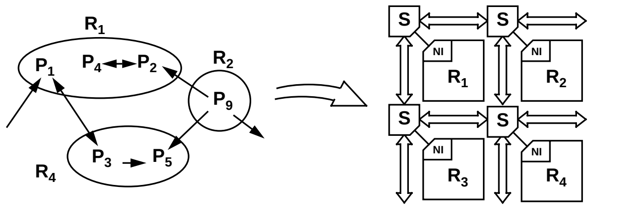

Fig. 2. *Nostrum* Process to Resource mapping

a structure where each switch is connected to its four switch neighbours and to its corresponding resource as depicted in Figure 2. From an implementation point of view, the resources (Processor cores, DSPs, Memories, I/O etc.) are the realisation of the *Processes (P)*. A resource can host single or multiple processes, potentially the processes can belong to one or several different applications. However, the *Nostrum* Concept is not inherently dependent of the mesh topology, other possibilities might include folded torus, fat-trees [14] etc. The reason why the mesh topology was chosen stems from reasons of three types.

First, higher order dimension topologies are hard to implement. As analysed by Culler [15], low dimension topologies are favoured when wiring and interconnects carry a significant cost, there is a high bandwidth between switches, and the delay caused by switches is comparable to the inter-switch delay. This is the case for VLSI implementations on the 2-dimensional surface of a chip and practically rules out higher dimension topologies. The torus topology was rejected in favour of a mesh since the folded torus has longer inter-switch delays.

Second, there is no real need for higher order dimension topologies. We assume that all applications we have in mind, e.g. telecom equipment and terminals, multi-media devices, and consumer electronics etc. exhibit a high degree of locality in the communication pattern. This is in stark contrast to the objective of traditional parallel computers; designed to minimise latency for arbitrary communication patterns.

Third, the mesh inhibits some desirable properties of its own, such as a very simple addressing scheme and multiple source-destination routes, which give robustness against network disturbances.

3 Theory of Operation

The switching of packets in *Nostrum* is based on the concept of deflective routing [17], which implies no explicit use of queues where packets can get reordered, i.e. packets will leave a switch in the same order that they entered it. This is possible since the packet duration is only one clock cycle, i.e. the length of packets is one *flit*. This means that packets entering a switch at the same clock cycle will suffer the same delay caused by switching and therefore leave the switch simultaneously. However, if datagram packets are transmitted over the network they may arrive in another order than they were sent in; since they can take different routes, this can result in different path lengths. The reason for packets taking different routes is that *the switching decision is made locally in the switches on a dynamic basis for every individual datagram that is routed through the network* – as stated earlier.

3.1. The Temporally Disjoint Networks

The deflective routing policy's non-reordering of packets creates an implicit time division multiplexing in the network. The result is called Temporally Disjoint Networks (TDNs). The reasons for getting these TDNs are *The Topology* of the network and *The Number of Buffer Stages* in the switches.

A. The Topology

Packets emitted on the same clock cycle can only collide, i.e. will only be 'in the same net', if they are on a multiple distance of the smallest round-trip delay. Intuitively this can be explained by colouring the nodes so that every second node is black and every second is white. Since all the white nodes are only connected to black nodes and all the black nodes are only connected to white nodes, any packet routed on the network will visit black and white nodes interchangeably. Naturally, this means that two packets residing in nodes of different colour, at a point in time, will never meet! That is, these two packets will never affect each others switching decisions. This is illustrated in Figure 3 (A); the network of a 4x4 mesh is unfolded and displayed as a bipartite graph in (B) where the left-side nodes only have contact with the right-side nodes and the opposite ditto. Please note that all the edges are bidirectional.

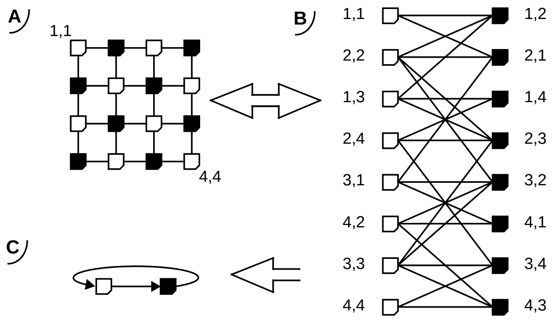

Fig. 3. Disjoint networks due to topology

This bipartite graph can further be collapsed into the lower left graph (C) of Figure 3 where all the black and white nodes are collapsed into one node respectively and the edges now are unidirectional. Logically packets residing in neighbouring time/space-slots could be seen as being in different networks, i.e. in Temporally Disjoint Networks. The contribution to the number of TDNs that stems from the topology is called the *Topology Factor.*

B. The number of buffer stages in the switches.

In the previous case where the topology gave rise to two disjoint nets, implicit buffering in the switches was assumed, i.e. a switching decision was taken every clock cycle. If more than one buffer is used in the switches, e.g. input and output buffering is used, this also creates a set of TDNs. In Figure 4, this is illustrated by taking the graph of

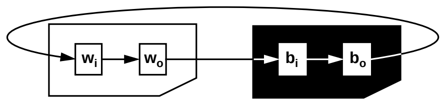

Fig. 4. Disjoint networks due to buffer stages in switches

Figure 3 (C) and equip it with buffers. The result is that every packet, routed on the network, must visit buffers in the following order: white input (w_i) -> white output (w_o) -

> black input (b_i) -> black output (b_o), before the cycle repeats. The result is a smallest round-trip delay of four clock cycles and hence four TDNs exist; where both the Topology Factor and the Buffer Stages contributes with a factor of two each. So in general

$$\text{TDN} = \text{Topology Factor} \times \text{Buffer Stages}$$

A clever policy when dealing with these multiple disjoint networks will give the user the option of implementing different priorities, traffic types, load conditions etc. in the different TDNs.

3.2. The Looped Container Virtual Circuit

Our Virtual Circuit is based on a concept that we call the *Looped Container.* The reason for this approach is that we must be able to guarantee bandwidth and latency for any VC that is set up. The idea is that a GB is created by having information loaded in a container packet that is looped between the source and the destination resource. The reason for this approach is the fact that it is very hard to guarantee admittance to the network at a given point in time as we shall see. This stems from two chosen policies

- Packets already out on the network have precedence over packets that are waiting to be launched out on the network.
- At a certain point in time the difference in the number of packets entering a switch, and the packets coming out after being switched, is always zero; that is, packets are neither created, stored, nor destroyed in the switches.

In Figure 5 (A), the consequence of these two policies is illustrated. The packet that wants to get out on the network never gets the chance since all the outgoing links are occupied. The switching policy, illustrated in Figure 5 (A), of letting the incoming packets be deflected, instead of properly routed, is not sufficiently for a proper network operation; but the sum of incoming/outgoing packets are the same, i.e. a deflected packet is occupying the same number of outputs as a packet routed to any other output!

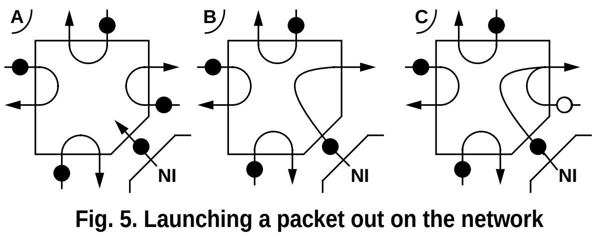

Fig. 5. Launching a packet out on the network

In Figure 5 (B), one link is unoccupied and the packet can therefore immediately get access to the network.

In Figure 5 (C), the principle behind our VC using containers as information carriers is illustrated. One ‘empty’ container arrives from the east, information from the resource is loaded, and the container is sent away.

In order to further illustrate the principle, Figure 6 depicts a VC going from the Source (1) to the Destination (3); a container belonging to this VC is tracked during four clock cycles. It is, in this example, assumed that the container already exists. In the first clock cycle, the container

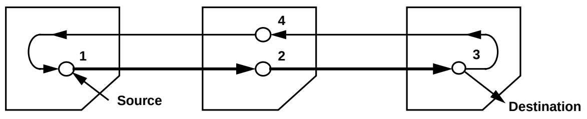

Fig. 6. The looped container

arrives to the switch connected to the Source. The container is loaded with information and sent off to the east. The reason why the information could be loaded instantly was that the container already was there and occupied one of the inputs. As a result of this, it is known that there will be an output available the following clock cycle.

In the second clock cycle, the container and its load is routed along its predefined path with precedence over the ordinary datagram packets originating from the BE traffic.

In the third cycle, the container reaches its destination, the information is unloaded and the container is sent back. Possibly with some new information loaded, but now with the original source as destination.

The fourth cycle is similar to the second.

3.3. Bandwidth Granularity of the Virtual Circuit

If the Looped Container and the Temporally Disjoint Networks (TDN) approaches are combined, we get a system where a limited set of VCs can share the same link. The number of simultaneous VCs, routed over a certain switch, is equal to the number of TDNs. This means that on-chip we can have many VCs, but only a limited set of VCs can be routed over the same switch – this since only one VC can subscribe to the same TDN on a switch output. To illustrate

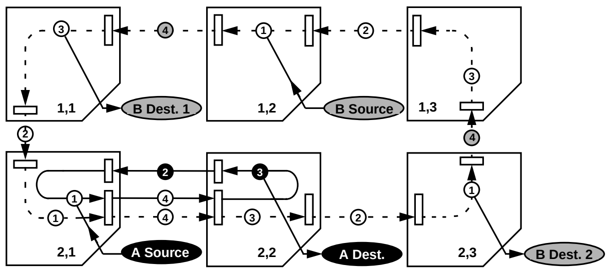

Fig. 7. BW granularity example

the concept, Figure 7 depicts two VCs; VC_A with black container packets and VC_B (path dashed) with grey ditto. In switch [2,1] and [2,2] the containers of both VCs will share the same links (and switch). The numbers inscribed in the packets, denotes which TDN the respective packet belong to; the numbers range from one to four since the number of TDNs, in Figure 7, is four since we have a bipartite topology and two buffer stages in every switch. As seen in Figure 7, VC_A have subscribed to TDN_2 and TDN_3, whereas VC_A only uses TDN_4.

The smallest bandwidth, the $BW_{Granularity}$, that is possible to acquire, for any VC, is dependent on the $VC_{Round\text{-}trip\ delay}$. The $VC_{Round\text{-}trip\ delay}$ is the length of the circular VC path in terms of buffers. In VC_A the $VC_{Round\text{-}trip\ delay}$ is four and in VC_B twelve. The $VC_{Round\text{-}trip\ delay}$ is the same as the number of containers a VC can have in all existing TDNs. Since the containers represent a fraction of the maximum BW over one link, the $BW_{Granularity}$ becomes

$$BW_{Granularity} = \frac{BW_{Max}}{VC_{Round\text{-}trip\ delay}}$$

The BW_{Max} is the switching frequency times the payload in the system, usually in terms of Gbit/s. The BW_{Max} that exist within one TDN is

$$BW_{Max(TDN)} = \frac{BW_{Max}}{TDN}$$

Of course several containers can be launched on a network if more than the initial $BW_{Granularity}$ is desired. The $BW_{Aquired}$ then naturally becomes

$$BW_{Acquired} = Container \times BW_{Granularity}$$

If the VC only subscribe to one TDN, the total number of containers is limited to

$$Container \le \frac{VC_{Round\text{-}trip\ delay}}{TDN}$$

Regarding the individual characteristics of VC_A and VC_B they are presented in Table 1.

	VC_A	VC_B
$BW_{Granularity}$ of BW_{Max}	1/4	1/12
Launched containers	2	2
Used TDNs	2	1
$BW_{Aquired}$ of BW_{Max}	1/2	1/6

Table 1. Summary of VC characteristics

4 Use of Concept

Accessing the VC is done from the NI. The set up of VCs is, in the current implementation, semi-static, this means that the route for the respective VC is decided at design time but the numbers of containers used by every VC is variable. That is – the bandwidth, for the different VCs, can be configured at start-up of the network. To set up the VC, i.e. to get the containers in the loop, the containers are launched during a start-up phase of the network where no ordinary datagram packets are allowed to enter the network. If more bandwidth is needed during run time, this can be achieved by launching more containers. However, in this case the set-up time can not be guaranteed since “new” container packets are not guaranteed access to the network. Naturally, if less bandwidth is needed some containers can be taken out of the loop.

Since the set-up of the VCs is based on a mutual agreement between the source and the destination regarding the information to be sent, no buffer overflow is assumed. That is, the source knows at what rate it can send data/packets to the NI and the destination knows what data rate it has to be able to cope with. If several applications reside in the same resource and need to be able to acquire bandwidth this could be handled by setting up several Virtual *Channels* residing in the same Virtual *Circuit*.

4.1. Multi-cast and other functionality

By the use of VC, several services, except for the obvious sending of data from a source to a destination at a guaranteed rate, can be implemented.

Multi-cast can easily be implemented by having multiple destinations along the VC path, as illustrated by VC_B of Figure 7, which has destinations in [1,1] and [2,3]. Even several source/destination pairs can be formed along a VC path subscribed to the same TDN as long as they are aligned so that the source is followed by the destination.

Even busses might be implemented quite effectively using the service of multi-cast. The sheer distribution of data is not of any problem but what might become a bottleneck is the bus master implementation. The delay/latency caused by the VC itself may reduce the bus master’s capability of granting/denying access to the bus due to latency. However, if latency is acceptable, nothing hinders an effective implementation of a bus structure.

4.2. Implementation

All services possible to implement using the VC container based concept, e.g. source – destination data distribution, multi-cast, or busses, utilises a combination of four standard switch functions

- **Source** Loads an incoming container with data from the appropriate NI output queue. Flags the packet as non-empty. Sends the container along the VC path
- **Destination (Final)** Read the data from the container and put it in the appropriate NI input queue. Flags the packet as empty. Sends the container along the VC path
- **Destination (Multi-cast)** Same as Destination (Final) but the container is not flagged as empty
- **Bypass** Sends the container along the VC path

Internally the VC path is handled by a small look-up table for every VC in the switch. In the current implementation, the VCs are set up semi-statically and the only extra HW needed in the switches is the one of giving a container packet the highest priority in the direction of its VC path. Also extra HW is needed to set/clear the empty bit dependent on the role of the switch (Source, Multi-cast Destination etc.) and whether to load/unload information. A switch with only BE functionality uses 13695 equivalent NAND gates for combinatorial logic (control), buffers excluded; for the same switch with the added functionality of VCs the gate count is 13896. So the relative extra HW cost is less than 2 percent! The number of gates is derived from Synopsys Design Compiler.

The additional cost, for implementing the VCs, in terms of bandwidth is very low; only two bits are used as packet header. The first bit identifies the packet as a container and the second flags the packet/container as empty or not. This means that the effective relative payload for a packet with 128 bits is more than 98 percent!

4.3. Simulation Results

Simulations carried out so far extend to HDL simulations with artificial, but relevant, workload models. The workload models used, implements a two-way process communication between A and B. In the first example AB uses BE for communication and in the second the VCs of the GB are employed. In both cases, the communication is disturbed by having random BE traffic in the rest of the network. As a vehicle for the simulation a 4X4 network was chosen. The processes were placed so that A got position [3,1] and B [2,4] in the 4X4 mesh. Both the background traffic as well as the traffic between A and B was created with the same probability, *p*. In the simulation *p* ranges from [0 .6], above that the network becomes congested due to fundamental limitations in capacity of the network.

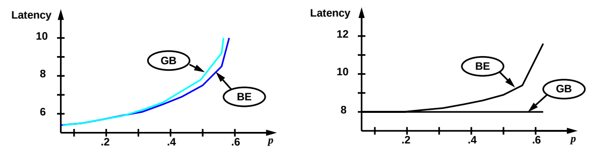

Fig. 8. The background traffic and the AB traffic

In Figure 8 the average latency is plotted against the probability of the packet generation, *p*. The left graph shows the background traffic and the right graph the AB traffic. BE and GB in the figure relates the respective graph to the traffic pattern used for AB traffic in the simulation.

As seen in Figure 8, the random background traffic in the network is very little affected by the VC; but for the AB traffic, the VC gives a tremendous boost in guaranteed latency and bandwidth for increased traffic in the network. The average bandwidth of the AB traffic is not changed – but now it is guaranteed! and as expected, the latency of AB goes from being exponential to become constant.

Of course, if more VCs were utilised, it would be theoretically possible to construct such traffic patterns and VC route mapping combinations so that network congestions are irreparable, but we found no interest in these artificial corner cases.

5 Conclusions

We have implemented a service of guaranteed bandwidth to be used in our NoC platform *Nostrum*. The GB uses Virtual Circuits to implement the two concepts that we call *Looped Containers* in *Temporally Disjoint Networks*. The VCs are set up semi-statically, i.e. the route is decided at design time, but the bandwidth is variable in run-time. The implementation of the concept was synthesised and simulated. The additional cost in HW, compared to the already existing BE traffic implementation and the cost in terms of header information were both less than 2 percent.

Simulations showed that the VCs did not affect BE traffic in the network significantly but gave a guaranteed bandwidth and a constant latency to the user of the GB. Also the cost of setting up the VC was very low.

Possible drawbacks are the potential waste of bandwidth in the returning phase of the container in the loop, since the container might travel empty if the BE traffic is one-way. Also, the limited granularity of bandwidth possible to subscribe to, might become a problem. Future work includes a method for clever traffic planning to avoid the possible waste of bandwidth when the VCs are set up.

REFERENCES

[1] Virtual Socket Interface Alliance, http://www.vsi.org

[2] Sonics Inc., http://www.sonicsinc.com

[3] L. Benini and G. DeMicheli, Networks on chip: A New SoC Paradigm. IEEE Computer, 35(1): p. 70 ff., January. 2002.

[4] P. Wielage and K. Goossens, Networks on Silicon: Blessing or Nightmare?. In Proc. of Euromicro Symposium on Digital System Design. Architectures, Methods and Tools, p 196-200, 2002

[5] P. Guerrier and A. Greiner, A Generic Architecture for On-Chip Packet-Switched Interconnections. In Proc. of DATE 2000, March 2000.

[6] W. J. Dally and B. Towles, Route packets, not Wires: On-Chip Inter-Connection Networks. In Proc. of DAC 2001, June 2001.

[7] M. Sgroi et al., Addressing the System-on-a-Chip Interconnect Woes through Communication-Based design. In Proc. of DAC 2001, June 2001.

[8] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K.Tiensyrjä, and A. Hemani, A Network-on-Chip Architecture and Design Methodology. In Proc. of IEEE Comp. Society, April 2002.

[9] K. Goossens et al., Networks on Silicon: Combining Best-Effort and Guaranteed Services, DATE 2002, March 2002.

[10] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch, Load Distribution with Proximity Congestion Awareness in a NoC, DATE 2003

[11] Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for NoC. E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, DATE 2003.

[12] M. Millberg, The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone, Technical Report TRITA-IMIT-LECS R 02:01, LECS, IMIT, KTH, Stockholm, Sweden, 2003.

[13] M. Millberg, E. Nilsson R. Thid and A. Jantsch, The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip, In Proc. of VLSI Design India, January 2004

[14] C. E. Leiserson, Fat Trees: Univ. Networks for Hardware Efficient Supercomputing. IEEE Computer, p 892 ff. vol. c-34, No 10, Oct. 1985.

[15] D. E. Culler and J. P. Singh, “Parallel Computer Architecture - a Hardware Software Approach”, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-343-3, 1999

[16] A Message Passing Interface Standard. http://www.mpi-forum.org

[17] U. Feige, P. Raghavan, Exact Analysis of Hot-Potato Routing. In Proc. of Foundations of Computer Science, p. 553 -562, 1992