A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping

Fernando Moraes, Aline Mello, Leandro Möller, Luciano Ost, Ney Calazans
Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)
Av. Ipiranga, 6681 - Prédio 30 / BLOCO 4 - 90619-900 - Porto Alegre – RS – BRASIL
{moraes, alinev, moller, ost, calazans}@inf.pucrs.br

Abstract

The increasing complexity of integrated circuits drives the research of new intra-chip interconnection architectures. A network-on-chip adapts concepts originated in the distributed systems and computer networks subject areas to connect IP cores in a structured and scalable way, pursuing the goal of achieving superior bandwidth to conventional intra-chip bus architectures. This paper presents the design of a switch targeted to a mesh interconnection topology. Each switch has 5 bi-directional ports, connecting 4 neighbor switches and a local IP core. They employ a XY routing algorithm, with input queue buffers. The main objective is to develop a switch with a small area, enabling its immediate practical use. The switch and a 2x2 mesh network were validated through functional simulation. Also the network has been successfully prototyped in hardware, using a million-gate FPGA.

Keywords*: network on chip, system on chip, core base design, switches, intra-chip interconnection.*

1. Introduction

Increasing transistor density, higher operating frequencies, short time-to-market and reduced product life cycle characterize today's semiconductor industry scenery [1]. Under these conditions, designers are developing ICs that integrate complex heterogeneous functional elements into a single chip, known as a System on a Chip (SoC). As described by Gupta et al. [2], SoC design is based on intellectual property (IP) cores reuse. Gupta et al. [2] define a core as pre-designed, pre-verified silicon circuit block that can be used to build a larger or more complex application on a semiconductor. These cores can be analog/digital cores, memory blocks, DSP cores, and also new technologies such as micro-electro-mechanical systems or optoelectronic systems [1][3]. Cores do not make up SoCs alone; they must include an interconnection architecture and interfaces to peripheral devices [3]. The interconnection architecture includes physical interfaces and communication mechanisms, which allow the communication between SoC components.

Usually, the interconnection architecture is based on dedicated wires or shared busses. The *dedicated wires* are effective for systems with a small number of cores, but the number of wires around the core increases as the system complexity grows up. Also, dedicated wires have poor reusability and do not offer flexibility. A *shared bus* is a set of wires shared by multiple cores. This approach is more flexible and is totally reusable, but it allows only one communication transaction at a time, because all cores share the bandwidth in the system and its scalability is limited to few IP cores, less than one or two dozens [4]. Using separate busses interconnected by bridges or hierarchical bus architectures may reduce some of these constraints, since different busses may account for different bandwidth needs, protocols and so on. Thus, each core can access the bus that fulfills its own requirements.

According to several authors [4]-[10], the interconnection architecture based on shared busses will not support the communication requirements for future ICs. According to ITRS, ICs will have billions of transistors, with feature sizes around 50 nm and clock frequencies around 10 GHz in 2012 [1]. In this context, a *Network on Chip* (NoC) appears as a possible solution for future on-chip interconnects. A NoC is an on-chip network [5] composed by cores connected to one switch, and switches connected by communication channels. This paper proposes a low area overhead NoC design based on a simple switch.

The rest of this paper is organized as follows. Section 2 presents an overview of the basic concepts and features of NoCs. The NoC communication protocol stack is discussed in Section 3. Section 4 details the main contribution of this work, the design of switch. The implementation and validation of a mesh topology NoC is described in Section 5. In Section 6, initial prototyping results are presented. Section 7 presents some conclusions and directions for future work.

2. Basic Concepts

As described in [9], NoCs are emerging as a possible solution to the existing interconnection architecture constraints, due to the following characteristics: (*i*) energy efficiency and reliability [7]; (*ii*) scalability of bandwidth when compared to traditional bus architectures; (*iii*) reusability; (*iv*) distributed routing decisions [6].

Two main parts compose a network, the *services* and the *communication system*. Rijpkema *et al* [9] describe some *services* considered essential for chip design, such

as data integrity, throughput and latency. The *communication system* supports the information transfer from a source to a target. As defined in [11], the communication among elements of a network is based on packet transference, which are often composed by a header, a payload, and a trailer. The NoC structure is basically a set of switches connected among them by communication channels. The way switches are connected define the *network topology*. Network topologies can be classified in two main classes: *static* and *dynamic* networks [12]. In *static* networks, each node has fixed point-to-point static connections to some number of other nodes. Hypercube, mesh, torus and fat-tree are examples of networks used to implement static networks. *Dynamic* networks employ dynamically configured switched channels. Busses and crossbar switches are examples of dynamic networks.

To ensure correct functionality in terms of message transfer, a NoC must avoid deadlock, livelock and starvation [11][12]. *Deadlock* may be defined as a cyclic dependency among nodes requiring access to a set of resources so that no forward progress can be made, no matter what sequence of events happen [12]. *Livelock* refers to packets circulating the network without ever making any progress towards their destination. It may be avoided with adaptive routing strategies. *Starvation* happens when a packet in a buffer requests an output channel and is blocked because the output channel is always allocated to another packet.

Besides avoiding the phenomena defined in the previous paragraph, it is necessary to specify how packets pass through the switches. Two methods for transferring messages are *circuit switching* and *packet switching* [13]. In *circuit switching*, a path is established before the packet is sent. When a circuit between source and destination has been established, the packet can be sent and any other communication on the allocated path is denied. In *packet switching*, a message is broken into packets that are transmitted through the network. The main difference between circuit and packet switching is that in the last one there is no communication channel reservation. In other words, the path is established dynamically. Packet switching implies the use of a *switching mode*. The most popular are *store-and-forward, virtual cut-through* and *wormhole* [13]. In *store-and-forward* mode, a switch cannot forward a packet until it has been received in its entirety. Each time a switch receives a packet; its contents are examined to decide what to do, implying per-switch latency. In *virtual cut-through* mode, a switch can forward a packet as soon as the next switch gives a guarantee that a packet will be completely accepted. Thus, it is necessary a buffer to hold a complete packet, like in store-and-forward, but in this case with lower latency communication. The *wormhole* routing mode is a variant of the virtual cut-through mode that avoids the need for large buffer spaces. A packet is transmitted between the switch in units called *flits* (flow control digits – the smallest unit of flow control). Only the header flit has the routing information. Then, the rest of the flits that compose a packet must follow the same path reserved for the header.

The *switching modes* define how packets move through switches. The *routing* defines the path taken by a packet between the source and the target. According to where routing decisions are taken it is possible to classify the routing in *source* and *distributed* [11]. In *source* routing, the whole path is decided at the source switch, while in *distributed* routing each switch receiving a packet decides where to send it. According to how a path is defined to transmit packets, routing can be classified as *deterministic* and *adaptive*. In *deterministic* routing, the path is uniquely defined by the source and target addresses. In *adaptive* routing the path is a function of the network traffic [11]. This last routing classification can be further divided into *partially* or *fully* adaptive. *Partially adaptive* routing uses only a subset of the available physical paths between source and target.

3. NoCs Protocol Stack

The OSI reference model is a hierarchical structure of seven layers that define the requirements for communication among processing elements [14][10]. Each layer offers a set of services to the upper layer, using functions available in the same layer and in the lower ones. NoCs usually implement a subset of the lower layers, such as Physical, Data Link, Network and Transport. These layers are described below in the NoC context.

The *physical layer* is responsible to provide mechanical and electrical media definitions to connect different entities at bit level [14]. In the present work this layer corresponds to the communication between switches, as exemplified in Figure 1. The physical data bus width must be chosen as a function of the available routing resources and available memory to implement buffering schemes.

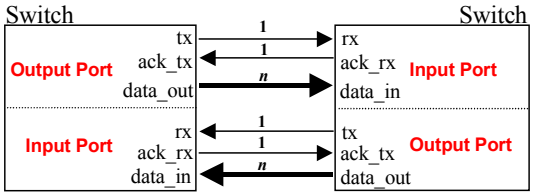


Figure 1 – Example of physical interface between switches.

The *data link* layer has the objective of establishing a logical connection between entities and converting an unreliable medium into a reliable one. To fulfill these requirements, techniques of flow control and error detection are commonly used [11]. This work implements in the data link layer a simple handshake protocol built on top of the physical layer, to deal with flow control and correctly sending and receiving data. In this protocol, when the switch needs to send data to a neighbor switch, it puts the data in the *data_out* signal and asserts the *tx* signal. Once the neighbor switch stores the data from the *data_in* signal, it asserts the *ack_rx* signal, and the transmission is complete.

The *network layer* is concerned with the exchange of packets. This layer is responsible for the segmentation and reassembly of flits, point-to-point routing between switches, and contention management. The network layer in this work implements the packet switching technique.

The *transport layer* is responsible to establish an end-to-end communication from source to target. Services like flow control, segmentation and reassembly of packets are essential to provide a reliable communication [11]. In this work, end-to-end connections are implemented in the IP cores connected to the NoC. The implementation of flow control and other options is envisaged as future work.

4. Proposed Switch

The main objective of an on-chip switch is the correct transfer of messages between IP cores. They usually have routing logic, arbitration logic and communication ports directed to other switches or cores. The communication ports include input and output channels, which can have buffers for temporary storage of information.

The switch proposed here has a routing controller logic and five bi-directional ports: East, West, North, South and Local. Each port has a buffer for temporary storage of information. The Local port establishes a communication between the switch and its local core. The other ports of the switch are connected to the neighbor switches, as presented in Figure 2. The routing controller logic implements the arbitration logic and a packet-switching algorithm.

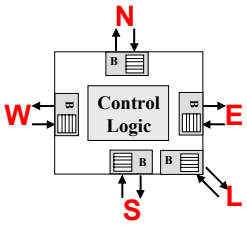


Figure 2 - Switch Architecture. B indicates input buffers.

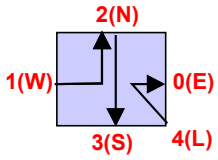
Among the switching modes presented in Section 2, wormhole was chosen because it requires less memory, provides low latency, and can multiplex a physical channel into more than one logical channel. Although the multiplexing of physical channels may increase the wormhole routing performance [13] this has not been implemented. The reason is to lower complexity and cost of the switch by using only one logical channel for each physical channel.

As previously described, the wormhole mode implies dividing packets into flits. The flit size is parameterizable, and the number of flits in a packet is fixed at $2^{(\text{flit size, in bits})}$. An 8-bit flit size was chosen here for prototyping and evaluation purpose. The first and the second flit of a packet are header information, being respectively the address of the target switch, named *header flit*, and the number of flits in the packet payload. Each switch must have a unique address in the network. To simplify routing on the network this address is expressed in XY coordinates, where X represents the horizontal position and Y the vertical position.

4.1. Control Logic

Two modules implement the control logic: *routing* and *arbitration*, as presented in Figure 4. When a switch receives a header flit, the arbitration is executed and if the incoming packet request is granted, an XY routing algorithm is executed to connect the input port data to the correct output port. The algorithm compares the actual switch address (xLyL) to the target switch address (xTyT) of the packet, stored in the header flit. Flits must be routed to the local port of the switch when the xLyL address of the actual switch is equal to the xTyT packet address. If this is not the case, the xT address is first compared to the xL (horizontal) address. Flits will be routed to the East port when xL<xT, to West when xL>xT and if xL=xT the header flit is already horizontally aligned. If this last condition is true, the yT (vertical) address is compared to the yL address. Flits will be routed to South when yL<yT, to North when yL>yT. If the chosen port is busy, the header flit as well as all subsequent flits of this packet will be blocked. The routing request for this packet will remain active until a connection is finally established in some future execution of the above procedure in this switch.

When the XY routing algorithm finds a free output port to use, the connection between the input port and the output port is established and the *in*, *out* and *free* switching vectors at the switching table are updated. The *in* vector connects an input port to an output port. The *out* vector connects an output port to an input port. The *free* vector is responsible to modify the output port state from free (1) to busy (0). Consider the North port in Figure 3(a). The output North port is busy (free=0) and is being driven by the West port (out=1). The input North port is driving the South port (in=3). The switching table structure contains redundant information about connections, but this organization is useful to enhance the routing algorithm efficiency.



(a)

	0 (E)	1 (W)	2 (N)	3 (S)	4 (L)
Free	0	1	0	0	1
In	-	2	3	-	0
Out	4	-	1	2	-

(b)

Figure 3 – Example of three simultaneous connections in the switch (a), and the respective switching table (b).

After all flits composing the packet have been routed, the connection must be closed. This could be done in two different ways: by a trailer, as described in Section 2, or using flit counters. A trailer would require one or more flits to be used as packet trailer and additional logic to detect the trailer would be needed. To simplify the design, the switch has five counters, one for each output port. The counter of a specific port is

initialized when the second flit of a packet arrives, indicating the number of flits composing the payload. The counter is decremented for each flit successfully sent. When the counter value reaches zero, the connection is closed and the *free* vector corresponding position of the output port goes to one (free=1), thus closing the connection.

A switch can simultaneously be requested to establish up to five connections. Arbitration logic is used to grant access to an output port when one or more input ports simultaneously require a connection. A dynamic arbitration scheme is used. The priority of a port is a function of the last port to have a routing request granted. For example, if the local input port (index 4) was the last to have a routing request granted, the East port (index 0) will have greater priority, being followed by the ports West, North, South and Local. This method guarantees that all input requests will be granted, preventing starvation to occur. The arbitration logic waits four clock cycles to treat a new routing request. This time is required for the switch to execute the routing algorithm. If a granted port fails to route the flit, the next input port requesting routing have its request granted, and the port having the routing request denied receives the lowest priority in the arbiter.

4.2. Message buffering

When a flit is blocked in a given switch, the performance of the network is affected, since the flits belonging to the same packet are blocked in other switches. To lessen the performance loss, a buffer is added to each input switch port, reducing the switches affected by the blocked flits. The inserted buffers work as circular FIFOs. The FIFO size is parameterizable, and a size eight has been used for prototyping purposes.

4.3. Switch Functional Validation

The proposed switch was described in VHDL and validated by functional simulation. Figure 4 presents the internal blocks of the switch and the signals of two ports (Local and East). Figure 5 presents a functional simulation for the most important signals of Figure 4.

The simulation steps are as follows (numbering below have correspondences in Figure 4 and in Figure 5):

1. The switch (xLyL=00) receives a flit by the Local port (index 4), signal rx is asserted and the data_in signal has the flit contents.
2. The flit is stored in the buffer and the ack_rx signal is asserted indicating that the flit was received.
3. The local port requests routing to the arbitration logic by asserting the h signal.
4. After selecting a port, the arbitration logic makes a request to the routing logic. This is accomplished by sending the header flit that is the switch target address (value 11) and the source of the input request (signal incoming, value 4, representing the local port) together with the request itself.
5. The XY routing algorithm is executed, the switching table is written, and the ack_rot signal is asserted indicating that the connection is established.
6. The arbitration logic informs the buffer that the connection was established and the flit can now be transmitted.
7. The switch asserts the tx signal of the selected output port and puts the flit in the data_out signal of this same port.
8. Once the ack_tx signal is asserted the flit is removed from the buffer and the next flit stored can be treated.
9. This second flit starts the counter indicating after how many clock cycles the connection must be closed (illustrated just in Figure 5).

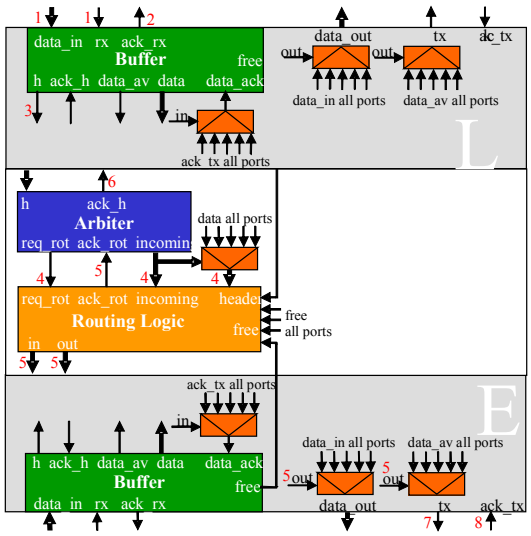


Figure 4 – Partial block diagram of the switch, showing two of the five ports. Numbers corresponds to the sequence of events in Figure 5.

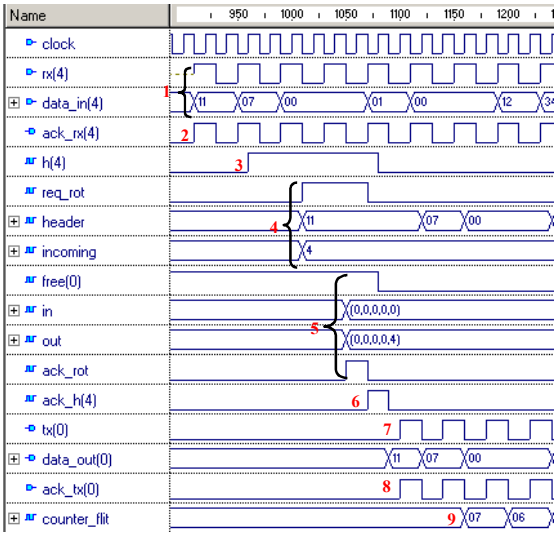


Figure 5 - Simulation of a connection between the Local port and the East port.

More complex validation procedures were also carried out but are not discussed for conciseness reasons. Text files containing packets are "connected" to the input ports. The received packets are written to text files, one for each port. It was verified that all sent packets were correctly received, validating the switch architecture.

5. Proposed Network On Chip

NoC topologies are defined by the connection structure of the switches. The proposed NoC assumes that each switch has a set of bi-directional ports linked to other switches and to an IP core. In the mesh topology used in this work, each switch has a different number of ports, depending on its position with regard to limits of the network, as shown in Figure 6.

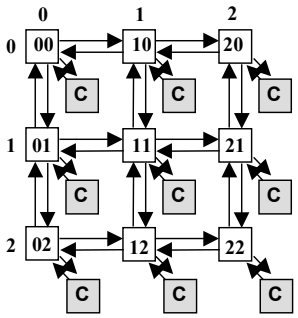


Figure 6 – 3x3 Mesh NoC structure. C marks IP cores, Switch addresses indicate the XY position in network.

The use of mesh topologies is justified to facilitate placement and routing tasks as stated before. The switch can also be used to build torus, hypercube or similar NoC topologies. However, building such topologies implies changes in switch connections and, more importantly, in the routing algorithm.

5.1. NoC Functional Validation

Packet transmission in the proposed NoC was validated first by functional simulation. Figure 7 illustrates the transmission of a packet from switch 00 to switch 11 in the topology of Figure 6. In fact, only the input and output interface behaviors of Switch 10 are shown in the simulation.

The simulation steps are as follows:

1. Switch 00 sends the first flit of the packet (address of the target switch) to the *data_out* signal at its East port and asserts the *tx* signal in this port.
2. Switch 10 detects the *rx* signal asserted in its West port and gets the flit in the *data_in* signal. It takes 10 clock cycles to route this packet, as explained in Section 4. Next flits are routed with a 2-clock cycle latency.
3. Thus, switch 10 output South port indicates its busy state in the free(3) signal. Signals free(i) are elements of the free vector defined in Section 4.1.
4. Switch 10 puts the flit in *data_out* signal and asserts the *tx* signal of its South port. Next, Switch 11 detects asserted the *rx* signal of its North port. The flit is captured in the *data_in* signal and the source to target connection is now established.
5. The second flit of the packet contains the number of flits composing the payload.
6. After all flits are sent, the connection is closed and the free tables entries of each switch involved in the connection return to their free state.

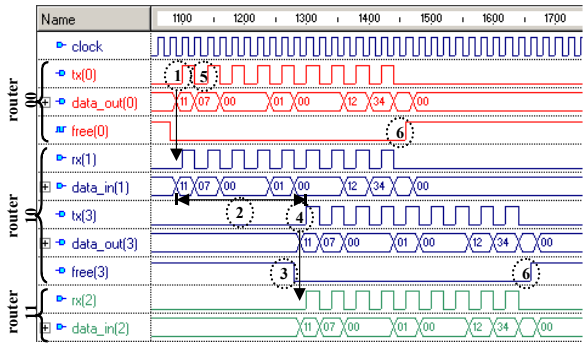


Figure 7 – Simulation of a packet transmission from switch 00 to switch 11 in the topology of Figure 6.

The minimal latency in clock cycles to transfer a packet from a source to a target switch is given by:

$$latency = \left(\sum_{i=1}^{n} R_i\right) + P \times 2$$

where: n is the number of switches in the communication path (source and target included), R_i is the time required by the routing algorithm at each switch (at least 10-clock cycles), P is the packet size. This number is multiplied by 2 because each flit requires 2-clock cycles to be sent.

6. Prototyping and Results

The proposed NoC was prototyped in the Memec Insight Virtex-II MB1000 development kit. This kit contains a 1-million equivalent gates Xilinx XC2V1000 456-pin FPGA [15]. A 2*x*2 mesh network was implemented. To validate the prototyped network two IP cores were developed: an RS-232 serial core and a memory core. The RS-232 serial core is responsible to send and receive packets to and from the network. The serial core was attached to Switch 00 and the memory cores were attached to the other three switches.

Two software programs were used for hardware validation. The authors implemented the first one to provide communication between the development kit and a host personal computer. The second software is ChipScope [15] from Xilinx. ChipScope allows visualizing FPGA internal signals selected by the user at run time.

The NoC with 4 IP cores (1 serial and 3 memories) and four switches was synthesized using the Leonardo synthesis tool. Table 1 presents the area data generated by synthesis, expressing the usage of FPGA resources. Approximately, 50% of the resources were employed, i.e. 500,000 equivalent gates.

Table 2 details the area usage of the NoC modules for two mappings, FPGA and ASIC. The switch itself takes 631 LUTs to be implemented, which represents 6,2% of the available LUTs in a million-gate device. The Table also gives area data for 3 simple IP cores: serial, memory, and an embedded small 16-bit processor (R8). The R8 processor is a 40-instruction, 16-bit non-pipelined, load store architecture, with a 16x16 bit register file. These cores are being used to build a NoC-based on-chip multiprocessing system. The SR (send/receive) is an adapter between the IP core and the communication interface. Additional glue logic is

needed to connect the IP core to SR, adding to the total gate count of the wrapped module.

Table 1 – 2x2 NoC area data for XC2V1000 FPGA. Gates are equivalent gates, LUTs are 4-input Look-Up-Tables, a slice has 2 LUTs and 2 flip-flops and BRAMs are 18-Kbit RAM blocks (flit width=8, buffer size=8).

Resources	Used	Available	Used/Total
Gates	513.107	1.000.000	51,31%
Slices	3.757	5.120	73,38%
LUTs	5.654	10.240	55,21%
Flip Flops	2.942	10.240	28,73%
BRAM	6	40	15,00%

Table 2 - NoC modules area report for FPGA and ASIC (0,35µm CMOS). LUTs represent combinational logic. ASIC mapping represents the number of equivalent gates (flit width=8, buffer size=8).

	Virtex II Mapping			ASIC
	LUTs	FFs	BRAM	Mapping
Switch	631	200	-	2.930
SR	193	233	-	1.986
Serial	91	93	-	752
Serial+ SR	590	563	-	4.587
RAM	-	-	2	-
R8	513	114	-	1.885
RAM + SR + R8	1.043	576	2	5.678

This multiprocessor NoC platform is presently used to execute parallel programs. The discussion of the results reported in this Section is presented next.

7. Conclusion and Future Work

The proposed switch and NoC fulfilled the requirement of implementing a low area overhead and low latency communication path for intra-chip modules.

It is already possible to compare area results obtained with some approaches found in the literature. First, Marescaux [16] employed exactly the same prototyping technology and proposed a switch that occupies 446 Virtex2 FPGA slices. The implemented switch employs 316 slices (631 LUTs) but it does not implement virtual channels. Second, the aSOC approach [17] mentions a switch ASIC implementation with an estimated transistor count of 50,000. The implemented switch with the smallest possible buffer size (since aSOC does not use buffers) and a 32-bit flit size (the same as aSOC) has an estimated gate count of 10,000, which translates to 40,000 transistors.

It is possible to note that the area of the IP cores is strongly influenced by the wrapper (SR module). The wrapper is still a preliminary structure, with buffers large enough to guarantee correct functionality of the communication. Better dimensioning of the wrapping structures is an ongoing work.

The most relevant point of this work is the availability of a hardware testbed where NoC architectures, topologies and algorithms can be implemented and evaluated.

The proposed NoC provides in its current state support to the implementation of *best effort* (BE) NoCs only [9]. In BE, sent the network can arbitrarily delay packets. For applications with real time constraints, it is necessary to provide *guaranteed throughput* (GT) services. Two techniques can be used to provide GT services, circuit switching and virtual channels.

Also, the IP core to NoC interface will be changed to a standard interface such as VSI or OCP, to provide reusability of the NoC modules.

8. References

[1] International Sematech. **International Technology Roadmap for Semiconductors - 2002 Update**, 2002. Available at http://public.itrs.net.

[2] Gupta R.K.; et al. **Introducing Core-Based System Design**. IEEE Design & Test of Computers, 14(**4**), Oct.-Dez. 1997, pp. 15-25.

[3] Martin, G.; Chang, H. **Tutorial – System on Chip Design**. In: ISIC´2001, Singapore, 2001.

[4] Kumar, S.; et al. **A network on chip architecture and design methodology**. In: ISVLSI´2002.

[5] Benini L.; al. **Networks on chips: a new SoC paradigm**. IEEE Computer, 35(**1**), Jan. 2002, pp. 70-78.

[6] Guerrier P.; et al. **A generic architecture for on-chip packet switched interconnections**. In DATE´2000.

[7] Benini, L.; et al. **Powering Networks on Chip**. In: ISSS´2001, pp. 33 –38.

[8] Dally, W.J.; Towles, B. **Route packets, not wires: on-chip interconnection networks**. In: DAC´2001, pp. 684-689.

[9] Rijpkema, E.; et al. **Trade Offs in the Design of a Router with both Guaranteed and Best-Effort Services for Networks On Chip.** In: DATE´2003.

[10] Sgroi, M.; Sheets, M.; Mihal, A.; Keutzer, K.; Malik, S.; Rabaey, J.; Sangiovanni-Vincentelli, A. **Addressing the system-on-a-chip interconnect woes through communication-based design**. In: DAC´2001, pp. 667-672.

[11] Duato, J. et al. **Interconnection Networks**. IEEE Computer Society Press Los Alamitos, CA. 1997, 515 p.

[12] Patterson, D.; Hennessy, J. L. **Computer Architecture: A Quantitative Approach**. Morgan Kaufmann, San Francisco, CA. 1996, 760 p.

[13] Mohapatra, P.; **Wormhole routing techniques for directly connected multicomputer systems,** ACM Computing Surveys, 30(**3**), Sep. 1998.

[14] Day, J.D.; Zimmermman, H. **The OSI reference model**. Proceedings of the IEEE, 71(**12**), Dec. 1983, pp. 1334-1340.

[15] Xilinx, Inc. **Virtex-II Platform FPGA User Guide**. Available at: http://www.xilinx.com (Jul. 2002).

[16] Marescaux, T.; Bartic, A.; Verkest, D.; Vernalde, S.; Lauwereins, R. **Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs**. In: FPL'02, Sep. 2002, pp. 795-805.

[17] Jian, L.; Swaminathan, S.; Tessier, R**. aSOC: A Scalable, Single-Chip communications Architecture**. In: IEEE International Conference on Parallel Architectures and Compilation Techniques, Oct. 2000, pp. 37-46.