VLSI IMPLEMENTATION OF A SWITCH FOR ON-CHIP NETWORKS

Tomas Henriksson, Daniel Wiklund, and Dake Liu
Dept. of Electrical Engineering
Linköpings universitet, 581 83 Sweden
E-mail: tomhe@isy.liu.se, Phone: +46-13-288956, Fax: +46-13-139282

Abstract. *Switch nodes in a 2D mesh SoC connection network have been suggested to solve the SoC integration problem. We have presented a memory buffer free switch node for connection circuits set up by packet routing. A test chip with two switch nodes has been implemented. The function of a switch node is to set up and tear down connections based on small packets and then to transport the payload data without buffering and with very low latency. The silicon cost of a node is 0.5 mm^2 based on a 2 metal layer 0.8 micrometer AMS CMOS technology. The connection latency cost is one clock cycle/switch node. The test chip works properly at 50 MHz.*

1. Introduction

The number of transistors that can be integrated on a single chip keeps increasing exponentially. This leads to the problem that it requires increasing effort to use all the available transistors in a useful way. The design productivity does not increase as fast as the manufacturing capability, the so called design gap. A proposed solution to this problem is to make use of reuse of large IP (intellectual property) blocks. These blocks can for example be complete processor cores, memories, configurable logic blocks or fixed function blocks.

The obvious problem that follows from the increased integration and reuse is that the IP blocks need to communicate. The traditional arbitration-based time-division multiplex (TDM) bus, e.g. the AMBA bus from ARM, Inc. [4], will no longer be able to keep up the pace when more masters share the bus. The main reason being that the TDM bus only allows one transfer at the time and thus has to arbitrate when there are multiple requests at the same time. The direct solution would be to break up the system in smaller domains that each use their own TDM bus and communicate with other domains via bridges between the local TDM buses. This will allow for a certain speedup in each domain but will severely limit the performance when communicating across domain borders. The flexibility of the system will also be impacted because the system has to be partitioned into the bus domains which will not allow for later changes in communication patterns.

We proposed the approach to use a switched on-chip network (OCN) as a replacement for the bus structure [1]. The OCN uses full crossbar switches with five ports, point-to-point links between the switches, and special format conversion blocks called wrappers to isolate the IP block port from the network port handling. We were later followed by other researchers suggesting similar solutions with switched networks for on-chip communication. The

common denominator in most suggestions including our solution is to connect switches in a two dimensional mesh network, see Fig. 1. Several recent suggestions use packet switching following the seven layer OSI model to isolate the different levels of protocols [5][6]. The seven layer OSI model is very suitable for the understanding of general purpose networks but is not suitable for direct implementation when it comes to on-chip communication like the one in an OCN. The principal reasons are that the OCN uses a topology that is fixed at design time and that the OCN always uses the same communication structure (or only a few variations) which means that many OSI layers can be collapsed in the implementation. Another possible drawback with several other suggestions is the packet switching which unfortunately leads to switches that either are more complex due to the use of virtual channels or that have to use buffers that can store entire packets in order to avoid situations like deadlock [7]. The packet latency in the OCN also becomes unpredictable and some packets can have significant latency (several hundreds of cycles) [8]. The packet-switched OCN is thus not suitable for hard real-time systems where tight deadlines must be kept at all times.

Instead of true packet switching we use small request packets to set up connection circuits. The request packets traverse the network in a similar fashion to a packet switched network. When the packet is routed through a node it locks the route which is subsequently used for the payload transfer that can be done without any buffering (except for a retiming register) in the switches. Just like the normal packet switching this scheme does not need any global arbiter or control which may significantly affect the performance. This novel hybrid packet/circuit switch solution is referred to as packet connected circuit (PCC) [3]. The PCC solution allows for very low latency in the switches, only five cycles for a route setup (request packet handling) and one cycle for payload transfer. The simplicity of the proposed routing scheme also allows for very high speed in the switches (at least 1.2 GHz in a 0.18 micron process) even with a standard cell implementation.

Our proposed system uses five port switches where four ports (north, east, south, and west) connect to the surrounding switches in the mesh and the fifth port (down) connects to the local IP block [2]. Since the network connections are fixed at the time of chip design, the switches can use a simple dynamic routing algorithm based on static knowledge of the direction to the destination (e.g. north-west) to select a free output link among the ones that leads closer to the destination. A round-robin scheme is used to select one output if there are several free outputs that lead closer to the destination.

The function of the OCN is to send data from the source wrapper to the destination wrapper via the shortest possible route. In this paper, we focus on building up and tearing down connections and on critical path analysis in the switch node. Additional functions such as the dynamic routing process are not described in this paper.

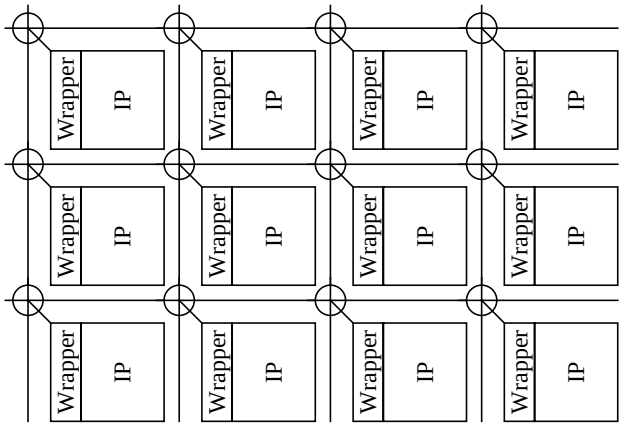


Figure 1: A two-dimensional switched on-chip network with wrappers and IP blocks.

In order to prove our concept of PCC switching a test chip with 2 simplified switches has been designed and manufactured. The aim of this paper is to describe that chip and thereby prove that the concept of circuit-switched OCNs is a viable solution for system-on-chip integration. In section II the simplifications from a real OCN are outlined and in section III the switch that has been implemented is described. In section IV the chip implementation is given in its detail. Section V presents our measured results and finally the conclusions are drawn in section VI.

2. Simplifications to the On-Chip Network

At the early stage in development when the testchip was built some simplifications to the switch were necessary in order to make it easily implementable. Since the implementation was done in a full custom flow instead of a standard cell flow the complexity had to be kept low so that the design time was not too long. The purpose of all simplifications was to decrease the design time but still retain the PCC functionality.

The first simplification was to limit the number of ports to three instead of the normal five ports. The second simplification was to just use four bits (a nibble) wide connections instead of the intended eight bits. These cuts in the switch specification do not affect the intended goal which was to prove the concept of PCC, it merely limits the number of concurrent data streams in the switch and the available bandwidth per stream.

The routing algorithm was also limited to a very simple source-addressed scheme where the source inserts a nibble with an output port number first in the stream for every switch that the route will traverse. Then each switch will consume the first nibble and use it as an output address. This scheme does not allow any dynamic routing since the source can not know the state of the switch at the time the stream will reach it. The payload is then sent directly following the last switch output address in the stream. The specifics of the stream format in this implementation are described in section 3.

The last simplification is that the implemented switch uses a fixed priority to select which input port is granted an output port in case of a collision (i.e. when two concurrent routing requests want to use the same output) in the switch.

The two last simplifications affect the routability of the OCN whenever there is more than one route active in the network. These simplifications make the OCN less efficient but does not affect the principles upon which the goal of the implementation relies.

3. The Switch

Since the switch that was implemented in the test chip has 3 ports it can support 3 contemporary data streams that flow through it. An overview of the switch is shown in Fig. 2. The function of a switch node is to pass a packet from the input port to one output port according to the address that the packet carries. When a core has data to send, it sends a request to the switch to which it is connected and then directly starts to send the data. If there is no possible route to the destination, the switch will send a negative acknowledge back to the core and the core has to retransmit later on. The switch makes use of a fixed non-preemptive priority scheme. That means that if a connection is set up it will last until the sender tears it down

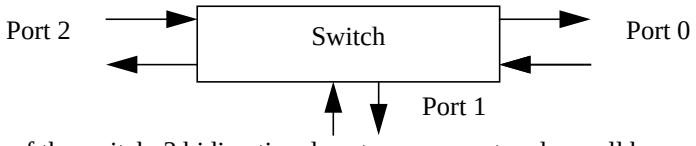


Figure 2: Overview of the switch. 3 bidirectional ports are present and can all be used simultaneously.

and that if two requests come to the switch at the same time always the request from the input port with the lowest port id will be serviced.

The ports consist of 4 data bits (a nibble) and 2 control signals. The data bits are used both for routing and for data. The 2 least significant data bits are used to route the packet. The input control signal, *req*, is used for connection setup, connection tear-down and the output control signal, *nack*, is used for negative acknowledgement.

3.1. Switching Functionality

Since the topology of the network is fixed at design time, each core has the knowledge of all the other cores on the network and how to route packets to them. A packet starts with the core pulling the *req* signal high and putting the first route address on the data bits, then consecutive route addresses are put on the data bits until the destination is reached. After the last address nibble, the data packet starts directly. For example if a core I is connected to switch A on port 0 and wants to send a packet to core II, which is connected to port 2 on switch B and switches A and B are connected as shown in Fig. 3. Then if core I wants to send a data packet to core II the start of the packet will look like the packet shown in Fig. 4. Switch A will receive the address 1 and pass on the packet to port 1, which goes to switch C. When doing that, the first address nibble will be removed from the packet and the *req* signal will be delayed by one clock cycle. So switch C will receive address 0 and route the packet to port 0 which goes to switch B. Also in switch C the first address nibble is removed and the *req* signal is again delayed by one clock cycle. Similarly switch B will receive address 2, route the packet to core II and remove the last address nibble. Thus, core II will receive the packet starting with the first data nibble, *d0*. While the data is sent, the connection stays up, but as soon as core I has sent the whole packet, the *req* signal is pulled low and the connection is torn down.

If a another connection request comes to port 2 of switch A with address 1 when the data transfer from core I is still active, that request will be given a negative acknowledgement because output port 1 is busy. If the request would have had address 0 on the other hand it would be connected to core I since it is only the input port 0 that is busy, not the output port 0, which is the one needed in this case.

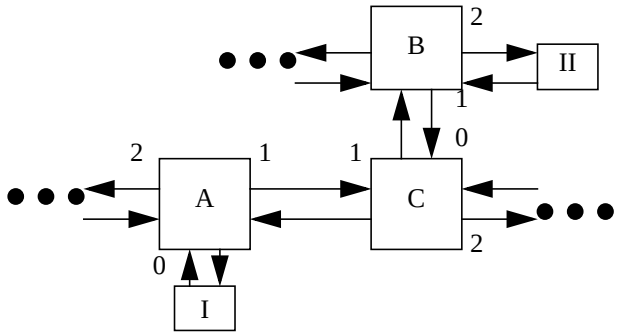


Figure 3: Example configuration of the network-on-chip. Only parts of the network are shown.

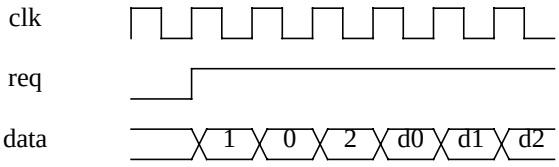


Figure 4: Packet for the example, core I sends data to core II.

3.2. Interior of the Switch

The switch is constructed from 4 control blocks and 1 crossbar. 3 control blocks are identical, one for each input port. The 4th control block controls the outputs, see Fig. 5. The input control blocks (ICBs) keep the connection information in a state register, see Fig. 6 (a). There is an edge detector in each ICB, which detects edges on the *req* signal. If there is a positive edge on the *req* signal, a connection request is sent from the ICB to the output control block (OCB). The decision logic in the ICB will then wait for a reply from the OCB and based on that reply it will either generate a negative acknowledgment, *nack*, or build up the connection by loading the state register with the address. When a connection is torn-down (i.e. there is a negative edge on the *req* signal), the ICB sends a tear-down order to the OCB and clears its state register. The decoder generates the crossbar control signals based on the value in the state register.

The structure of the OCB is shown in Fig. 6 (b). When there are multiple requests for the same output at the same clock cycle, the input port with the lowest id number has the highest

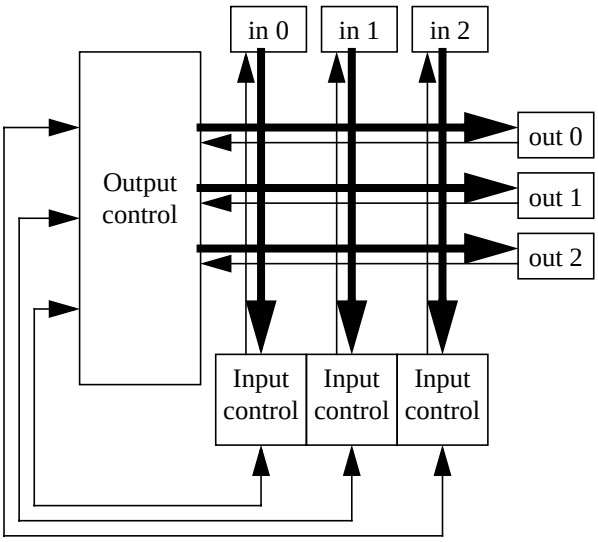
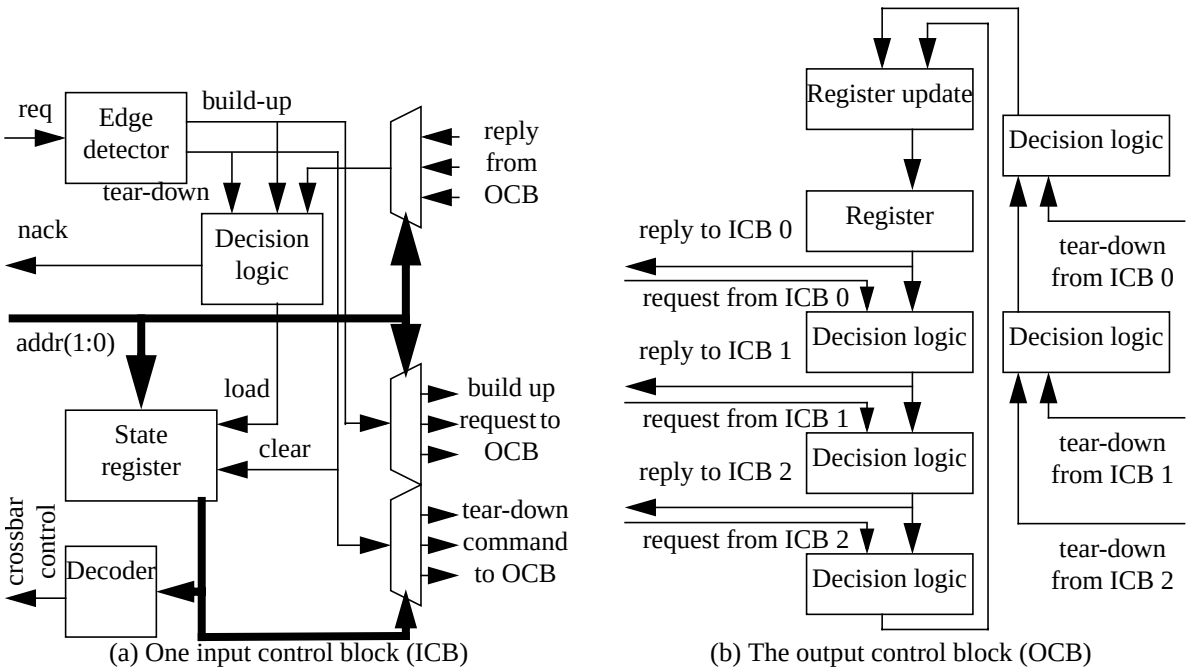


Figure 5: Block diagram of the switch.



(a) One input control block (ICB)

(b) The output control block (OCB)

Figure 6: The control blocks

priority. For example, if ICB 1 and ICB 2 try to build up connections at the same time to output port 0. Then the status of output port 0 passes through the first decision logic block (DLB) unchanged, since ICB 0 did not request output port 0. ICB 1 will thus get a positive reply from the OCB for the request on port 0. In the second DLB the status will however change since there is a request from ICB 1. So ICB 2 will get a negative reply and generate a *nack*. In the third DLB the status will pass through unchanged, since output port 0 is already taken. In the register update block, the new status will be computed based on the result of all connection build-up request and connection tear-down orders.

The crossbar is built up from 9 identical cells. The structure of a cell is shown in Fig. 7. The 3x3 crossbar is controlled by the crossbar control signals from the state register decoders in the ICBs. Each ICB generates three control signals, one for each of the three crossbar cells on its input port.

4. Chip Implementation

The chip that was designed consists of two switches. There are also two test sequence generators (TSGs) on chip. The TSGs were included in the design because of the pin limitation (the design was limited to one corner of a chip). Only one output port and one input port from each switch could be accessed from the pins. The other input ports are used for connection between the switches and for the TSGs. One output port on each switch is used for the connection between the switches and the third output port is left unused, see Fig. 8.

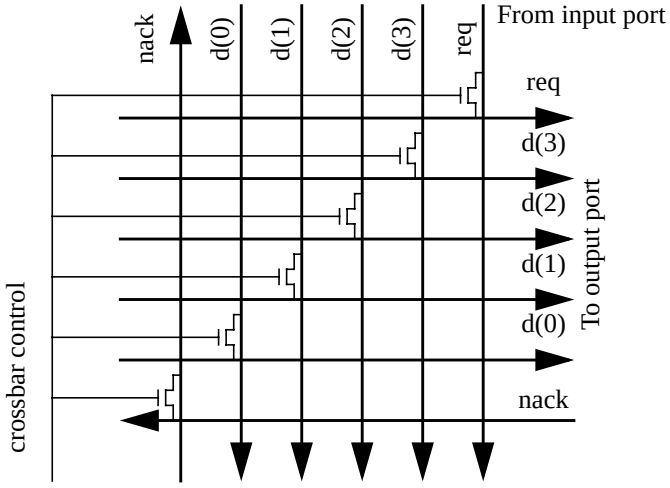


Figure 7: Structure of a crossbar cell.

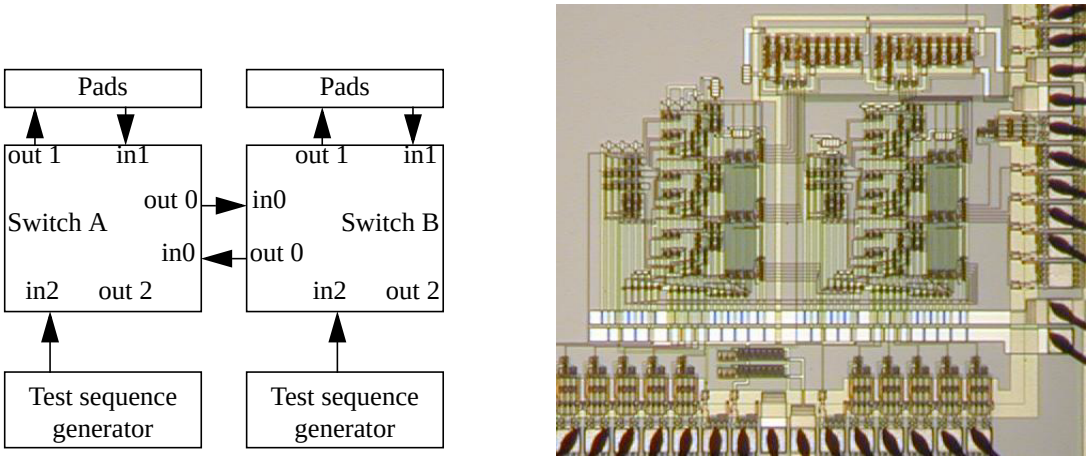


Figure 8: Chip overview. Block diagram (left) and chip photo (right)

The TSGs are triggered by a start signal and then they generate a packet which is addressed to port 1 and then contains the sequence 0x1, 0xD, 0xA, 0x8, 0x4 and 0x2 on the 6 following nibbles. This sequence is routed to the output port 1 which goes to the pads. The TSGs are used in the testing phase to test the priority of the switches.

The chip was designed using a full-custom design technique in a two metal layer, 0.8 micron technology from AMS. The schematic and the layout were designed in Mentor Graphics LED and simulations were performed by LSIM and Spice.

All the four blocks in this chip run from the same external clock with little skew, so the whole chip is synchronous in its operation.

5. Results and Discussion

The chip was manufactured and tested in a HP16500 logic analysis system from HP/Agilent. The chip photo is shown in Fig. 8. In the upper part, the two TSGs can be seen and below them there are the two switches. The area of one switch node is 0.5 mm^2. All pads are not shown on the chip photo.

One of the pads was not bonded, as can be seen on the photo (the third from below on the right pad row). This pad is connected to the *req* signal to input port 1 of switch A. We therefore cannot use that input port, instead we had to connect the unbonded pad to ground by a special probe. However, all major functions of the switch could still be tested by using the other input ports and the other switch.

The test results proved functional correctness for all the planned test cases. Those test cases were:

- Simple connection build up and tear down
- Test sequences from TSGs to outputs
- Cascaded routing from input port 1 on switch B to output port 0 and continuing to input port 0 on switch A to output port 1
- The non-preemptiveness of an already existing connection
- The priority mechanism for simultaneous connection requests

The logic analysis system that we used can only have a clock period of multiples of 10 ns. The chip works correctly for 50 MHz clock, but not for 100 MHz clock. The exact maximum performance of the switch is somewhere in between. The limitation could also be the board that was used for testing. Since our main interest was not the performance but the functionality we did not spend time and money on a printed circuit board. Instead we used an experimental board with hand soldered wires to connect the logic analyzer and the pattern generator with the test chip. The power consumption for the chip at 5 MHz and 50 MHz clock frequency at 5 V supply voltage is shown in table 1. The core includes the two switches and the two TSGs.

The switch that was designed is of course a simplification of a switch for a real OCN. In a real OCN, the switches will not be synchronous, but can be assumed to run on phase-shifted clocks from one single source and therefore capturing the input data is relatively straight-forward. The interfaces to the cores can be converted by a wrapper, which adjusts bus width and switching frequency of the signals to fit the core.

A switch in a real OCN will also have more than 3 ports and support more complex routing protocols. Nonetheless our switch implementation has proved that the concept of circuit switched OCNs where the connections are built up by the headers of the packets is working.

A switch in a real OCN will be designed in a much more modern process technology and thereby the performance can be increased a lot, even if a standard cell based design flow is used.

The critical path in the switch is situated in the OCB, in the DLBs that handle the requests from the ILBs. This path has a delay which is linear to the number of ports in the switch. When implementing a switch with 5 ports the critical path will therefore be longer. There are however ways to get around that problem. All the DLBs could be merged into one combinational logic block with several inputs. This is easy to do when using HDL descriptions and synthesis to standard cells. In this full custom implementation we chose to use one small DLB cell that we could replicate many times to save design effort. Another way to speed up the OCB would be to use pipelining technique in the decision chain. By doing so the critical path is relaxed to only one DLB, but the cost is that the latency until a reply is generated equals the number of pipeline stages in the OCB.

There is a lot of related work in this area concerning the topology and functionality of the OCN, as was discussed in the introduction section. However we think that this is the first actual chip implementation to be published.

Table 1: Power consumption

Clock frequency	Core power consumption	Pad frame power consumption
5 MHz	0.95 mW	3.55 mW
50 MHz	3.20 mW	29.85 mW

6. Conclusions

A fully functional, yet simplified, switch for an OCN has been designed, manufactured and tested. This proves that circuit-switch based OCNs is a viable solution to the SoC integration problem.

Future work includes the design and manufacturing of second generation test chip, which will incorporate switches with all the features discussed in this paper, a larger 2-D mesh network and real-time traffic sources.

Acknowledgments

This study is part of the SoCBUS project, which is sponsored by Swedish Foundation for Strategic Research (SSF).

References

[1] Daniel Wiklund and Dake Liu, “Switched interconnect for system-on-a-chip designs”, *Proceedings of the IP 2000 conference*, Edinburgh, Scotland, Oct 2000, pp 185-192

[2] Daniel Wiklund and Dake Liu, “Design of a System-on-Chip Switched Network and its Design Support”, *Proceedings of the International conference on communications, circuits and systems (ICCCAS)*, Chengdu, China, June 2002, pp 1279-1283

[3] Dake Liu, Daniel Wiklund, Erik Svensson, Olle Seger, and Sumant Sathe, “SoCBUS: The solution of high communication bandwidth on chip and short TTM”, *Proceedings of the Real time and embedded computing conference*, Gothenburg, Sweden, Sept 2002.

[4] ARM, Inc., “AMBA 2.0 specification”, http://www.arm.com

[5] William J. Dally and Brian Towles, “Route packets, not wires: On-chip interconnection networks”, *Proc of the Design Automation Conference (DAC)*, Las Vegas, USA, 2001, pp 684-689

[6] Iikka Saastamoinen, David Sigüenza-Tortosa, and Jari Nurmi, “Interconnect {IP} node for future System-on-Chip Designs”, *IEEE int’l workshop on Electronic design, Test, and Applications (DELTA)*, 2002, pp 116-120

[7] Christopher J. Glass and Lionel M. Ni, “Fault-tolerant wormhole routing in meshes without virtual channels”, *IEEE transactions on parallel and distributed systems*, vol 7, no 6, 1996, pp 620-636

[8] Pierre Guerrier and Alain Greiner, “A Generic Architecture for On-Chip Packet-Switched Interconnections”, *Proc of the design and test in Europe (DATE)*, Munich, Germany, 1999, pp 250-256