High-level Analysis for Reconfiguration of a Fault Tolerant Mesh-based NoC Architecture Using Transaction Level Modeling

Homa Alemzadeh[1], Fatemeh Refan[1], Paolo Prinetto[2], Zainalabedin Navabi[1]

[1] *CAD Research Laboratory*
Department of Electrical and Computer Engineering
School of Engineering, University of Tehran
Tehran,Iran
{homa,refan}@cad.ece.ut.ac.ir, navabi@ece.neu.edu

[2] *Politecnico di Torino*
Dipartimento di Automatica e Informatica
Torino, Italy
Paolo.Prinetto@polito.it

Abstract

In this paper we propose a fault-tolerant architecture for mesh-based network on chips to recover from permanent faults in switches. We add a number of spare links between processing elements and their neighboring switches and some logic redundancy to network elements. Also online fault detection and correction strategies are presented for this architecture. We model and simulate the communication behavior of the proposed architecture and recovery methods using TLM. Fast simulation speed of TLM as a high-level description language helps us in easy design exploration of our fault-tolerant architecture and finding the most efficient architecture for it. Later in the design process, it is possible to refine this TLM model to lower abstraction levels and synthesize it to an actual hardware.

Index Terms *— Fault-tolerant Architecture, Networks on Chip, Transaction Level Modeling*

1. Introduction

With the advance of the semiconductor technology, and the increase in complexity of integrated circuits, networks on a chip are becoming the main solution for addressing the communication challenges in System-on-Chip architectures. Using NoCs leads to more performance improvement than traditional communication structures in SoCs like bus-based communication and dedicated point-to-point links [1].

On the other hand, Transaction Level Modeling (TLM) is regarded today as the next step in the direction of complex integrated circuits and systems design entry. Design, modeling, and simulation in higher levels of abstraction like TLM provide designers with faster simulation. This helps them in early design space exploration, task partitioning and better testbench developing leading to a shorter time to market. Contrary to this movement to TLM abstraction level for digital system design, most common approaches for fault modeling and simulation, test methodologies and fault tolerance approaches are still at the RT and gate level. Therefore, an important need for high-level design methodologies is developing methods for high level testing and repair of digital systems.

Most of the researches in the field of fault-tolerant NoC architectures concentrate on tolerating transient errors by adding information redundancy like coding methods. Adding hardware redundancy with error detection is another method for fault-tolerant that is used less in this area. [2] presents a self-repair method based on using redundant links and cross-points to increase yield and reliability for NoC interconnects. [3] takes advantage of both methods of information and hardware redundancy for tolerating transient, permanent and intermediate faults. [4] proposes a semi-regular mesh based NoC architecture, where without changing the regular structure of an NoC a few long range spare links are added to improve the performance. Although these spare links have been used with the aim of performance improvement in NoC, but they can be useful as redundant links to increase reliability as well.

The fault tolerant mesh-based NoC architecture we present here recovers from a single switch fault by adding a spare link between each processing element and one of its neighboring switches. In this architecture we do not require any repeater modules, as those used in [4]. Also contrary to [2] we propose fault detection/correction strategies accompanying with the hardware redundancy used in our architecture. We take advantage of TLM SystemC 2.0 [5] as a high-level hardware description language focusing on separation of computation and communication parts for modeling and fast simulation of our NoC architecture. High-level NoC model presented here has been cut down by hiding computation details and just modeling the communication behavior of the NoC system.

The rest of paper is organized as follows. In Section 2 we introduce the basic NoC architecture model we used in our work, and will present an overview on transaction level modeling. Section 3 describes the proposed fault tolerant NoC architecture. We present our TLM simulation methodology and experimental results in Sections 4 and 5 respectively. Finally we conclude our paper in Section 6.

2. Background

In this section we present the simplified model of NoC architecture used in the analysis and methods of this paper. In addition, a brief overview of TLM as our modeling scheme for NoCs will be presented here.

2.1. NoC Architecture

The primary NoC topology which is used in this paper is a simple regular mesh-based architecture. Figure 1.a shows a 3×3 version of this architecture, where PE_i is the i^{th} processing element and SW_i is the switch directly connected to it.

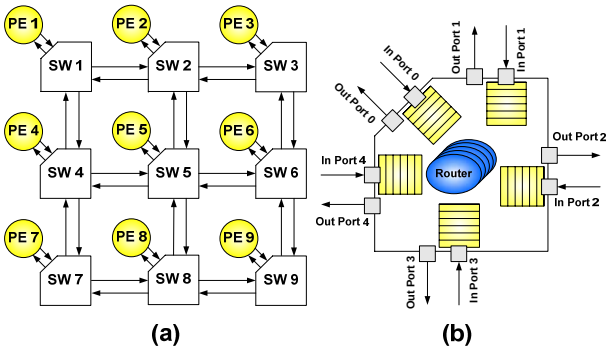

Figure 1. (a) A 3×3 mesh-based NoC architecture, (b) Structure of a NoC switch

In this architecture each switch consists of two parts: a communication section composed of five identical input/output ports and a routing logic as is shown in Figure 1.b. All the switches have one port connected to their corresponding processing element and the others to neighboring switches. Each port is a bidirectional link, where the input link goes through a circular FIFO as buffer for storing the incoming packets.

The packet structure is defined based on the communication protocol, the necessary features like QoS, and the application mapped onto the NoC architecture. We have considered only the most common fields, including a header that determines the source and destination addresses, and a data payload containing some information about the path the packets pass through. When a packet arrives on one of the input ports of a switch, it is stored in the input FIFO waiting for the router to send it to the proper output port. A round robin algorithm determines the next packet to be routed, and the output port is selected using X-Y routing algorithm.

Each processing element in this NoC consists of a processing unit and a network interface. The network interface part is responsible for communicating with the switch directly connected to the processing element by formatting the output packets.

2.2. Transaction Level Modeling

With the increase in complexity of digital systems and shrinking time to market, Electronic System Level (ESL) design has become a design methodology for implementing large digital systems. ESL industry has introduced transaction level modeling (TLM) as the next step in the direction of system level design entry.

TLM is a transaction-based modeling approach founded on high-level programming languages such as SystemC. It highlights the concept of separating communication from computation within a system. In the TLM notion, components are modeled as modules with a set of concurrent processes that calculate and represent their behavior. These modules exchange communication in the form of transactions through an abstract channel. Based on the degree of modeling accuracy, we have two fundamental classes of TLM: Un-timed TLM, also known as *programmer's view* (PV) and timed TLM that is given another name as *programmer's view plus timing* (PVT) [6].

Since starting the design process with a very high-level model like un-timed TLM ignores many details and timing annotations in design specification, the number of events that must be processed by a simulator decreases dramatically. This results in a faster simulation than lower modeling levels like RT and gate level. Therefore designers can decide on problems like module partitioning, system functionality, and developing test benches in a more organized fashion at early stages of design.

3. Fault-Tolerant NoC Architecture

In this section we discuss the problem we are addressing and our proposed solution. First our proposed fault-tolerant architecture is introduced and then an appropriate method for online fault detection and recovery in this architecture is presented.

3.1. Problem Statement

In the regular mesh-based NoC architecture presented in Section 2, each processing element is connected to a single switch via a single link. If this link or the whole switch becomes faulty, then the entire system would face two problems. First, the switch's directly connected processing element becomes inaccessible, because its only communication link to the rest of network is broken. The second problem is that the faulty switch cannot be used anymore for routing and passing packets, therefore routing algorithm should be changed accordingly to bypass it.

In order to solve these problems, we propose a fault tolerant NoC architecture in which a spare interconnect is added between each PE and one of its neighboring switches. Figure 2 shows all possible spare links for *PE5* that can be added to the regular 3×3 mesh-based NoC when *SW5* becomes faulty.

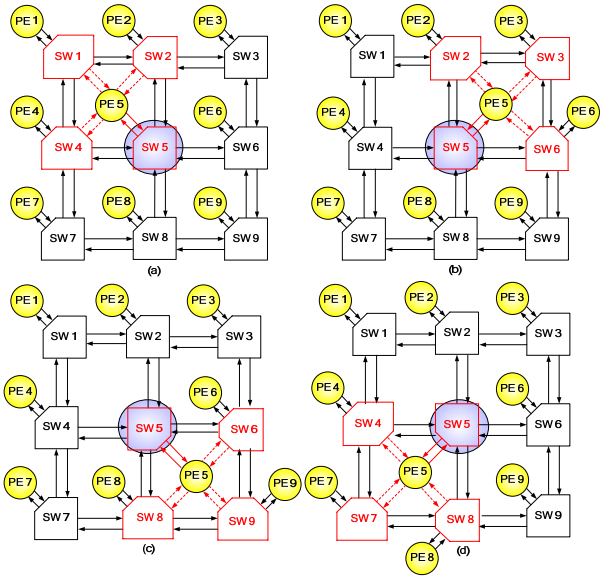

Figure 2. Possible spare links for *PE5* when *SW5* becomes faulty

Although adding all the possible spare links to all processing elements increases the overall reliability, but also changes the overall NoC topology and leads to a high cost in terms of hardware redundancy. Adding spare links to a PE require additional in/out ports and input FIFOs for the processing element for each of the corresponding switches. Also some logic should be added to the network interface part of the PE to handle multiple spare links connected to it. In order to avoid this cost in our architecture, we just choose one of the possible spare links for a processing element.

The best spare link to be added to a processing element is chosen among different possibilities based on a performance analysis that we perform in our high-level NoC simulation model.

In this architecture a simple logic is added to the network interface of each PE. This logic stores the ID of the added spare link and sends packets via this new link in the case of fault detection. Each switch has an embedded BIST module responsible for continuously testing its operation and reporting its status by setting a *fault_status* flag. Storing the address of the chosen alternative switch and logic for changing the routing algorithm for bypassing the faulty switches is also necessary in each switch. In addition, we need two new control fields for each packet: an *n*-bit field (in a *n*×*n* NoC) named *FSN* for keeping the location of the faulty switch in the NoC, and a flag called *CR* for indicating cases in which the routing should be changed.

3.2. Online Fault Detection and Recovery

This work focuses on permanent single faults occurring in NoC switches. The embedded BIST structure in each switch continuously updates its *fault_status* to report the status of the switch. Neighboring switches and processing elements check this flag before starting any communication with the switch. A switch, even in the case of being faulty, should be capable of informing its neighbors of its faulty status and sends them the ID of the switch used as its alternative.

Consider switch *i* in Figure 3 as a faulty switch detected in the NoC. Say, for example, that we choose to connect its PE to SW_{i-1} with a spare link. When PE_i or each of *j*, *i+1*, *k*, and *i-1* switches intend to send a packet to *i*, they check its *fault_status* flag to become sure that this switch works properly. Finding out that switch SW_i is faulty, PE_i and the neighboring switches (*j*, *i+1*, *k*, and *i-1*) arrange different strategies to recover from this fault.

PE_i sends its packets through SW_{i-1} instead of SW_i as shown in path (a) of Figure 3. The network interface part of PE_i sets the value of *FSN* field of packets to *i*, the ID of the faulty switch, and if the destination is *i*, it changes it to *i-1*, the ID of the alternative switch.

In addition to PE_i the neighboring switches should also change their routing strategy to avoid passing packets through the faulty switch. When each of of the neighboring switches, *j*, *i+1*, *k*, or *i-1*, are to send a packet to switch *i*, and detect that it is faulty, they change the routing for sending the packet to this switch. This re-routing strategy takes place based on

the destination of the outgoing packet. First of all the *FSN* field of the packet is set to the ID of faulty switch, and the destination field is checked; if the destination is *i*, it is changed to *i-1*. Also if the selected output port is horizontal (i.e., via ports 2 or 4), the packet is sent to one of the vertical output ports 1 or 3 based on the destination column. However, if the selected output port causes a vertical movement (i.e., via ports 1 or 3), the *CR* field of the packet is set to change the routing strategy from X-Y to Y-X algorithm. Then the packet is sent to one of the horizontal output ports 2 or 4 according to its destination. When a switch receives a packet with the *CR* field set to Y-X, it routs the packet using the Y-X algorithm instead of the default X-Y.

Switches used as alternative switches also need their own recovery strategies for dealing with a faulty switch.. When a packet, with destination *i-1* reaches the alternative switch *i-1*, it should be sent to PE_i or PE_{i-1} based on the *FSN* field. If *FSN* is equal to *i* the packet is routed to the new connected PE_i instead of PE_{i-1}.

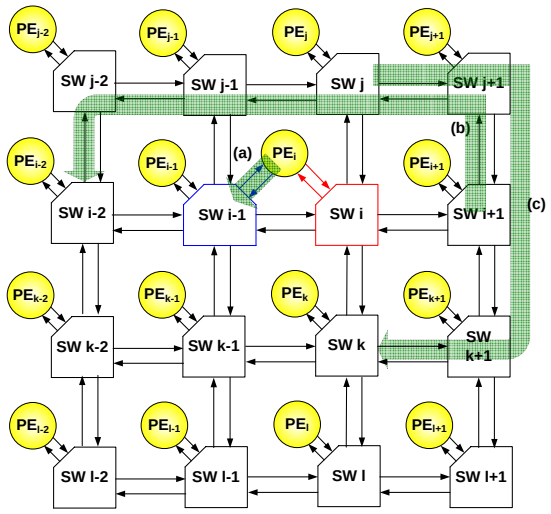

Figure 3. Fault detected in SW_i, (a) PE_i spare link to alternative switch, (b) Rerouting path for a horizontal case, (c) Rerouting path for a vertical case

As stated before, X-Y is the default routing algorithm, and we choose Y-X as an alternative method in vertical movements in the network. In case of a fault in a neighboring switch, if the output port is horizontal, then according to X-Y, we have three situations based on the row of the destination PE. If the destination is in a row above the current position, the packet is sent to port 1, and if it is in a switch below the current one, then the packet is routed to output port 3. Otherwise this row is the destination row and the packet is arbitrarily sent to one of the vertical ports to bypass the faulty switch. In any of the three cases stated above, the packet goes to the next row and the X-Y routing algorithm will not let the packets to return back to this row, except in the case that the destination is the faulty switch and the spare link is attached to a switch in this row. This procedure avoids any deadlocks in the routing. For example, consider the case in which a packet is to send to PE_{i-2} from switch *i+1*. Even this case will not cause a deadlock since if switch *j+1* is chosen as the next destination instead of *i*, the packet traverses switches *j+1*, *j*, *j-1*, *j-2*, and *i-2* according to the X-Y algorithm. See path (b) in Figure 3.

Similarly, in the case of confronting a faulty switch via vertical output ports, after changing the destination field, if needed, the packet should be re-routed. If the destination is in a column to right or left of the current column, the packet is sent to output ports 2 or 4 respectively. But this will cause a deadlock in the case that the destination is in the same column of the source switch. This is because (based on the X-Y algorithm), the next switch will send back the packet to the source in order to pass it vertically through the same column. As an example see the highlighted path (c) in Figure 3. If *j* tries to send a packet to *k*, then according to the recovery strategy it sends the packet to switch *j+1* instead of *i*, if the routing algorithm remains X-Y, *j+1* will then send the packet back to *j* and again encounter to faulty switch SW_i. This will continue forever, leading to a deadlock situation. The *CR* field we introduced in our strategy is for preventing this situation by telling the next switches to route this packet by the Y-X algorithm instead of X-Y.

The proposed routing/re-routing method can be implemented by look up tables for X-Y and Y-X algorithms similar to what is presented in [7] with some additional logic for switching between them.

4. Simulation Method

This section shows how we find the best fault tolerant NoC architecture for a specific input application mapped onto an NoC. A scheduled and bounded task graph of the application mapped onto the NoC architecture is used as input and faults are injected into different switches of this structure. Then we simulate different spare link selection possibilities in the presence of each fault, and compute the worst-case delay of system in each case. This way, the best spare links in terms of the minimum worst-case delay to be added to the fault-tolerant NoC architecture are chosen for the input application.

We have implemented the NoC architecture presented in Section 3 in SystemC TLM 2.0 library. Switches and processing elements are modeled as

SystemC modules with *SC_THREAD* processes implementing their behavior. Links between the modules that NoC packets travel through are modeled by *tlm_fifo* channels. These channels are for modeling unidirectional data transfers. We take advantage of TLM approximate-time abstraction level for modeling and performance evaluation of our system. Therefore, without considering timing details and just by using a number of *wait()*statements we can model the traffic seen by each packet in every switch in high-level.

Furthermore, since we concentrate on the faults in switches as communication elements of NoC and consider spare links for recovery from these faults, the critical bottleneck would be the communication performance of system after applying our healing method. Therefore we just focus on the communication behavior of the NoC architecture and ignore the computation details of processing elements and switches in our model. TLM features for separation of communication and computation parts within a system help us in reaching this goal.

At the start of simulation, each processing element distinguishes its own tasks and stores the information about its entering and outgoing packets from a specific input application defined by a *communication task graph* (CTG) like what is presented in [8]. A CTG is a directed acyclic graph, where each vertex represents a computational module of the application referred to as a task [9]. The edges represent the communication with adjacent nodes, and the weight on each edge is equal to the communication volume between tasks. This representation characterizes the partitioning, task assignment, scheduling, communication patterns, and task execution time of an application [8]. The execution of the mapped input application onto the NoC architecture is simulated based on the task orderings and timings derived from this input CTG.

We consider the worst case delay of the system as the evaluation parameter for our proposed architecture. After arrival of each packet to its destination, its delay is calculated based on the timing information it carries inside. This timing information includes the number of passed switches, inter-switch links, network interface to switch wrappers, processing elements seen, and the turns wasted waiting in the input FIFOs of switches. Since we have cut down our model and do not consider detailed timing annotations at the lower abstraction levels, just a rough estimate of delay which represents relative values is sufficient for comparison of results. So the delay for each switch is calculated as a weighted sum of the above timing parameters. Also in order to ease the process of monitoring packets traveling through the network, each source PE sends an *end-data* packet after each transmission of a burst data. The overall worst-case delay of the system is estimated by the maximum delays of *end-data* packets arriving at the last destination.

5. Experimental Results

In this section we present our simulation results for a complex multimedia application (MMS). The input CTG used in our simulation is shown in Figure 4. This is the communication task graph of a complicated MMS application, including an MP3 and H263 encoder and decoder [10]. In [10] this system is partitioned into 40 tasks, scheduled and then bounded to 16 distinct IPs including ASIC, Memory, CPU, and DSP cores. The scheduled and bounded CTG has been greedily mapped onto our 4×4 mesh-based NoC architecture as is shown in Figure 4. Then we simulate this mapped application using the TLM simulation method presented earlier. Figure 5 shows the simulation results for injecting faults into switches 1 to 5 in the 4×4 NoC. The overall worst-case delay of

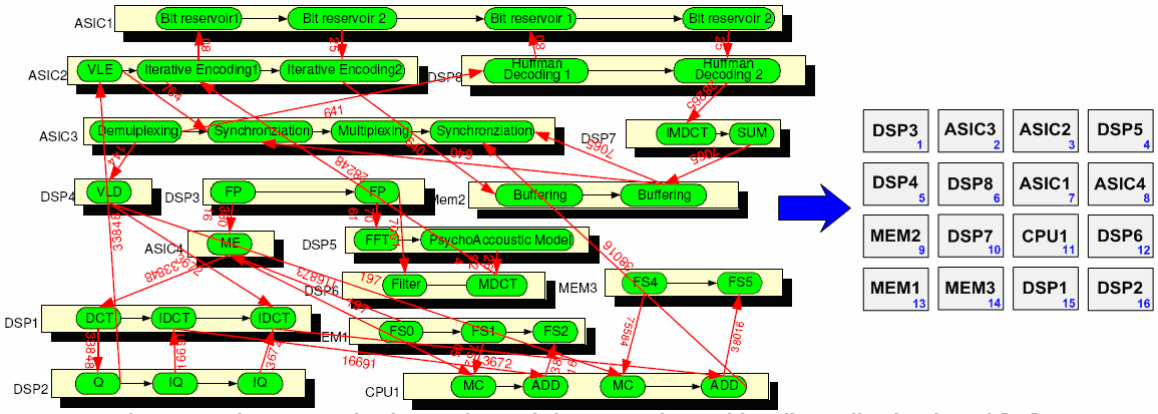

Figure 4. Left: Communication Task Graph for a complex multimedia application (MMS) [10], Right: Greedily mapped onto our 4×4 mesh-based NoC architecture

the system in the case of choosing each of the possible alternative switches is shown in this chart.

The first column shows the worst-case delay in the presence of no faults. We can see that applying our proposed fault recovery strategy can improve the performance compared to the normal operation mode when no faults exist in the system. This shows that this architecture can also be useful as a performance improvement solution for decreasing traffic in an NoC.

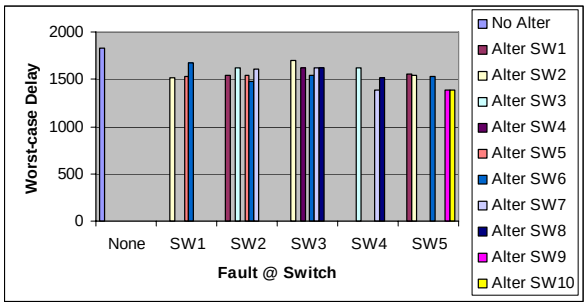

Figure 5. Simulation Results for CTG of Figure 4, Worst-case delay with different alternative switches

As an example of a faulty switch consider the case that *SW1* becomes faulty. There are three possibilities for *PE1* to continue its work by using a spare link to one of the alternative switches *SW2*, *SW5* or *SW6*. The second column in the chart shows the corresponding worst-case delays for each of these choices. Choosing *SW6* as the alternative switch results in the worst delay, while choosing *SW2* gives minimum delay. We can verify this result by considering our input CTG. As shown in Figure 4 the IP core mapped on *PE1* is *DSP3*. This figure shows that *DSP3* just communicates with *ASIC4* and *DSP5* mapped onto *PE4* and *PE8* respectively. It is clear that using *SW2* as the spare link for *PE1*is the best choice, because *SW2* is the first switch in the paths to *PE4* and *PE8* with an X-Y routing algorithm. Connecting *PE1* to *SW2* is even better than the normal mode in which *PE1*sends its packets through *SW1*.

6. Conclusions

The proposed architecture has several advantages. First of all the layout implementation is not needed to be changed completely, and contrary to what is presented in [4] no additional repeaters is needed for the added links. Next, by adding these spare links to the NoC, its architecture still remains almost regular, this has a minor effect on the fabrication process.

Furthermore, as a future work we can extend the proposed architecture to work in two modes: normal and faulty. In the normal mode we can take advantage of spare links as a solution for directing traffic from busy switches to idle ones and improve the system performance. While in faulty mode they act as described in this work for bypassing faulty switches and improve the reliability.

On the other hand, the TLM-based simulation method presented here is helpful for easy modeling and fast exploration of NoC architectures. We take advantage of HW/SW co-design capabilities of TLM for simulating the execution of a specific input application mapped onto NoC and analyzing its performance factors, as fast as a software program. Also the high level hardware description of the architecture can be refined to lower TLM abstraction levels to RT level SystemC, which makes it possible to be synthesized into its hardware.

7. References

[1] T. Bjerregaard and S. Mahadevan. “A survey of research and practices of network-on-chip”, ACM Computing Surveys, 38(1), 2006.

[2] C. Grecu, A. Ivanov, R. Saleh, and P. P. Pande, “NoC interconnect yield improvement using crosspoint redundancy”, in *Proceedings of the 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT ’06)*, pp. 457–465, Arlington, Va, USA, October 2006.

[3] T. Lehtonen, P. Liljeberg and J. Plosila, “On-line Reconfigurable Self-Timed Links for Fault Tolerant NoC”, *VLSI Design*, Journal of Hindawi Publishing Corporation, Apr. 2007.

[4] Umit Y. Ogras, Radu Marculescu. “It’s a small world after all: NoC Performance Optimization via Long-range Link Insertion”, *IEEE Trans. on Very Large Scale Integration Systems, Special Section on Hardware/Software Codesign and System Synthesis*, 14(7):693-706, July 2006.

[5] OSCI SystemC TLM 2.0 Standard, *http://www.systemc.org/projects/tlm/document/TLM_2.0_Overview/en/l*

[6] Frank Ghenassia (editor), *Transaction-level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*, Springer, 2005.

[7] Ali Shahabi, Nima Honarmand, Hassan Sohofi and Zainalabedin Navabi. “Degradable mesh-based on-chip networks using programmable routing tables”, *IEICE Electron. Express*, Vol. 4, No. 10, pp.332-339, (2007).

[8] Jingcao Hu, Radu Marculescu. “Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints”, Asia & South Pacific Design Automation Conference (ASP-DAC), January, 2003.

[9] J. Hu, R. Marculescu, “Communication and Task Scheduling of Application-Specific Networks-on-Chip”, IEEE Proceedings Computers & Digital Techniques, Sep. 2005.

[10] Jingcao Hu. “Design Methodologies for Application Specific Networks-on-Chip”, PhD thesis, Carnegie Mellon University, May, 2005.