Reliability in Application Specific Mesh-based NoC Architectures

Fatemeh Refan[1], Homa Alemzadeh[1], Saeed Safari[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
{refan, homa, safari, navabi}@cad.ece.ut.ac.ir

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

Abstract

Networks on Chips (NoCs) provide a mechanism for handling complex communications in the next generation of integrated circuits. At the same time, lower yield in nano-technology, makes self repair communication channels a necessity in design of digital systems. This paper proposes a reliable NoC architecture based on specific application mapped onto an NoC. This architecture is capable of recovering from permanent switch failures via replacing them by neighboring switches. This method has hardware and power consumption overhead, but significantly improves reliability and has a very little effect on the performance of the system. We suggest a reliability analysis method based on the combinatorial reliability models and use it to evaluate our proposed fault-tolerant NoC architecture.

1. Introduction

By increasing the complexity of integrated circuits, NoCs are becoming the main solution for addressing the communication challenges in SoC architectures. Using NoCs leads to more performance improvement than the traditional communication structures [1]. On the other hand, this fast scaling in technology has caused rapid increase of the probability of facing faults in different NoC components [2].

Two kinds of errors may generally occur in an NoC: *Transient* and *Permanent* errors. Recovery techniques have been proposed for each kind of these errors. The most common transient error recovery technique is retransmission mechanism following error detection techniques like coding [3]. On the other hand, most researches targeting at permanent error recovery use dynamic rerouting to bypass the failed hardware. Several rerouting schemes have been proposed that either update the routing tables [3], [4], [5], and/or add extra information to the packets aiming at fault tolerant routing strategy. The other important self-healing method is based on hardware redundancy, and spare insertion. Method presented in [6] is a self-repair method based on using redundant links and cross-points to increase yield and reliability for NoC interconnects. In [7] authors take advantage of both information and hardware redundancy for tolerating transient, permanent and intermediate faults. In [8] performance has been improved using long-range links which can be applied for link and router failure tolerance too.

The main problem with the current techniques is that if a switch fails, considering mesh architecture, recovery cannot be accomplished using rerouting only. In addition hardware redundancy is needed to repair the lost connection to the network of the processing element directly connected to the failed switch.

The next issue to consider is the architecture evaluation metric. Fault-tolerant architectures can be evaluated from reliability and performance aspects. Most related researches have used performance as their evaluation factor [3], [4], [5]. Reliability can be evaluated analytically or using simulation. E.g. [9] has evaluated reliability of an NoC with dynamic power management as a function of time by means of simulation.

In this paper we present a fault tolerant mesh-based NoC architecture with the ability of recovering from single permanent switch failures by adding a redundant link between each processing element and one of its neighboring switches. This architecture does not require any repeater modules, as those used in [8]. Also contrary to [6], here fault detection/correction strategies are accompanying the hardware redundancy we add to the architecture. This is done by proposing a deterministic low-cost routing/re-routing strategy. We also estimate the improvement in reliability of presented architecture using an application-specific routing-based analytical model.

2. Background

This section presents background information for understanding the rest of paper. The NoC model and application mapping onto NoCs are discussed.

978-0-7695-3264-6/08 $25.00 © 2008 IEEE
DOI 10.1109/IOLTS.2008.53

IEEE computer society

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 our architecture. In this architecture each switch consists of five identical input/output ports and a routing logic as shown in Figure 1.b. Each port is a bidirectional link with a circular FIFO on its input side.

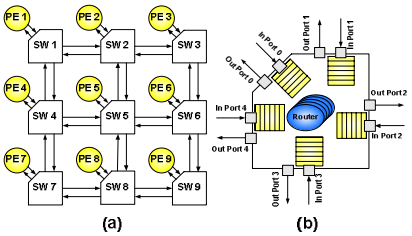


Figure 1 (a) A 3×3 Mesh-based NoC, (b) NoC Switches

We have considered only the most common fields for the structure of NoC packets. This includes a header that determines the identification number, source, and destination addresses, and payload containing data in addition to some information about the path the packets pass through. Packets are selected based on round-robin algorithm spanning various switch FIFOs in sequence and are routed using the *XY* 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 PE by processing and formatting the input and output packets.

2.2. Application Mapping onto an NoC

An application mapped onto an NoC can be represented as a Communication Task Graph (CTG). A CTG is a directed acyclic graph which represents computational modules (tasks) of the application and the volume of communication between them. For modeling and simulation of running an input application on a pre-selected NoC architecture, we should first schedule the application tasks represented by a CTG and bind them onto the corresponding processing elements of the target NoC [10].

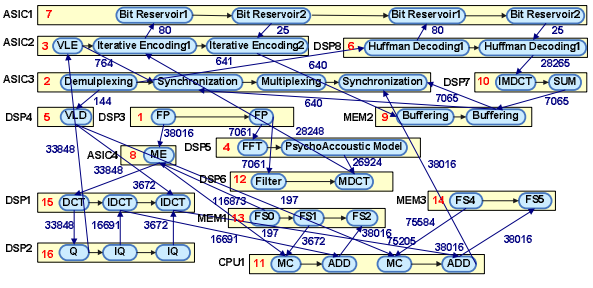


Figure 2 CTG of a Complex Multimedia Application (MMS) [10]

Figure 2 shows a CTG of a complicated MMS application, including an MP3-H263 encoder/decoder. This system is partitioned, scheduled and bounded to 16 distinct ASIC, Memory, CPU, and DSP cores [10].

3. Fault Tolerant NoC Architecture

In the NoC model presented in Section 2, each processing element is connected to a single switch. If a switch failure occurs in this architecture, the entire system would face two problems: First, the switch's directly connected processing element becomes inaccessible; Second, the failed switch cannot be used anymore for routing and passing packets.

The fault-tolerant NoC architecture we propose here recovers from these switch failures by making modifications to the original NoC.

3.1. NoC Architecture Modifications

In our new NoC architecture a spare interconnect is added between each PE and one of its neighboring switches. This architecture prevents a processing element from becoming inaccessible when its switch fails.

For each processing element there is more than one neighboring switch to be connected to with a spare link (e.g. *PE5* of Figure 1 has 8 choices: Switches 1, 2, 3, 4, 6, 7, 8, and 9). To maintain the NoC architecture still regular and minimize the hardware cost, we only choose one of the possible spare links for a processing element. The best spare link for each PE is chosen among different possibilities based on an application specific performance analysis performed before programming the NoC. The target fault-tolerant architecture is then configured and programmed based on this selection done for all the processing elements.

3.2. NoC Components Modifications

The proposed fault tolerant architecture in addition to the discussed spare links, introduces some modifications in the NoC processing elements, switches, and packets:

- **Processing elements;** A simple logic is added to the network interface of each PE, storing the ID of its alternative switch.
- **Switches;** Proper logic is needed to guarantee that a switch, even when faulty, is capable of informing its neighbors of its fault status. Storing the address of the chosen alternative switch and logic for changing the routing algorithm for bypassing the switches is also necessary in each switch.
- **Packets;** Two new control fields for each packet are necessary: an *n*-bit field (in a *n*×*n* NoC) named *FSN* (Faulty Switch Number) for keeping the location of the faulty switch in the NoC, and a flag called *CR* (Change Routing) for indicating cases in which the routing should be changed.

4. Online Fault Recovery

We assume that fault detection is done by self testing facilities embedded in each switch responsible for continuously testing its operation. A switch is made capable of informing its neighbors of its faulty status by setting a *fault_status* flag and sending them the *ID* of the switch used as its alternative. This flag is checked by the neighboring switches and processing elements before starting any communication with the switch.

The re-routing strategy proposed here uses three fields of a packet: *DST*, *FSN*, and *CR*. The value of the *DST* field always shows the *ID* of the destination switch. *FSN* field is set to the *ID* of the destination PE when the packet is re-routed. Finally, the *CR* field indicates whether the routing should be *XY* or *YX*. The default routing mechanism is *XY*, but it is changed to *YX* to prevent deadlock in some situations explained later.

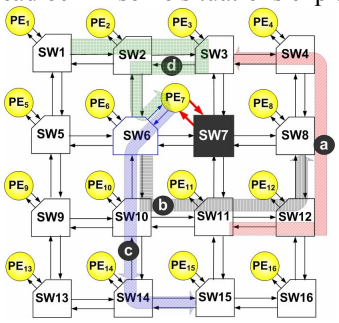


Figure 3 Failure of SW7: (a) Rerouting path for a vertical case, (b) Rerouting path for a horizontal case, (c) PE7 as Source, (d) PE7 as Destination

Consider switch *7* of Figure 3 as an example of a faulty switch detected in the NoC. If we choose to connect its PE to SW_6 with a spare link, PE_6 is called the local PE, and PE_7 the spare PE of SW_6. When PE_7 or each of SW_3, SW_6, SW_8, SW_{11} intend to send a packet to SW_7, they check its *fault_status* flag to make sure that this switch works properly. Finding out that switch SW_7 is faulty, the *FSN* field of the packet is set to the *ID* of faulty switch, and the destination is changed to *6* if it is PE_7. Also if the selected output port is horizontal (vertical) ports 2 or 4 (1 or 3), the packet is sent to one of the vertical (horizontal) output ports 1 or 3 (2 or 4); we call this situation horizontal (vertical) pass. Furthermore in the vertical pass the *CR* field of the packet is set to change the routing strategy from *XY* to *YX* algorithm. The selection of the new output port is done on the basis of the destination switch *ID*. In the case of horizontal (vertical) pass, we have two situations:

1. If the destination is in the same row (column) as the current, output port 1(4) or port 3(2) are chosen with the probability of $p_u(p_l)$ and $p_d(p_r)$, respectively, where $p_u(p_l) = 1\text{-}p_d(p_r)$. For example paths (a) and (b) of Figure 3 demonstrate the right and down paths of a vertical and a horizontal pass respectively.
2. If the destination has a different row (column) from the row (column) of the current switch, the output port is chosen based on the destination row (column). If the destination row (column) is above (in left hand of) the current row (column), port 1(4) and in the other case port 3(2) is chosen. Paths (c) and (d) of Figure 3 are two examples of this situation where the source and destination switch are faulty, respectively.

In situation 2 of the vertical pass (e.g. Path d), re-routing packet based on *XY* algorithm causes that the next switch in the adjacent column (*SW2*) send back the packet to the sender switch (*SW3*) because the sender is in the column of destination (*SW7*). This will continue forever, leading to a deadlock situation. The *CR* field is for preventing this situation by telling the next switches to route the packet according to *YX* algorithm instead of *XY*.

Using the above strategy, always the shortest path is chosen, the packet moves towards the destination in each step, and the faulty switch is encountered only once, thus avoiding deadlocks.

When a packet arrives at the destination switch it is sent to the local PE or the spare one based on the *FSN* field. If *FSN* is set, the packet is routed to the spare PE, otherwise the local PE is the destination.

The proposed reconfiguration capabilities will be implemented as additional look-up tables and logic inside the NoC switches and processing elements.

5. Spare Link Selection

This section shows the method we use for spare link selection to find the best fault-tolerant configuration for an application specific NoC. The best spare links are selected based on a high-level simulation before programming the NoC. The scheduled and bounded CTG of the application to be mapped onto NoC is taken as input. Then assuming that there can only be a single permanent failure in the switches, we simulate the selection of different possible alternative switches for each PE and select the ones which lead to the best overall performance in terms of the average response time of the system.

The NoC simulation model is implemented using SystemC TLM 2.0 [11] library. Since the critical bottleneck of a system is the communication performance, we ignore the precise functionalities and computation details of the processing elements in our model. The worst case response time of the destination processing elements and the average response time of the system are considered as the evaluation parameters. Since our

models are at the transactional level, and do not consider detailed timing annotations at the lower abstraction levels, only a rough estimation of the delay is sufficient for comparison of the results. The delay for each packet is calculated after its arrival to the destination based on the timing information it carries including the number of passed switches, inter-switch links and network interface to switch wrappers, processing elements observed, and the turns wasted waiting in the input FIFOs of the switches as shown in Eq. 1.

$$Packet\_Delay = t_{LINK} \times n_{LINK} + t_{NI} \times n_{NI} + t_{PE} \times n_{PE} + t_{FIFO} \times n_{FIFO} + t_{SW} \times n_{SW} \quad \text{Eq. 1}$$

The average response time of the system is estimated by the maximum delays of last packets arriving at the destination processing elements.

Following this simulation, we need to choose only one spare link for each PE. While the ideal alternative switch for several PEs may be the same, each switch is limited to having only one spare link connected to it. To solve this problem we use an exhaustive search algorithm to explore the space of different spare link selection possibilities for the PEs of an NoC. We choose the best alternative switches for all the PEs for the input application based on the average response time of the system.

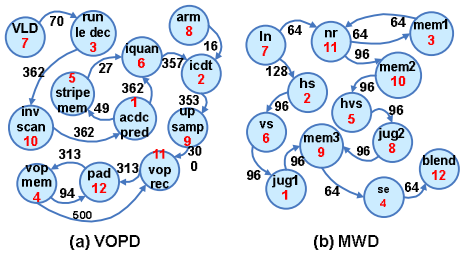


Figure 4 CTGs of VOPD and MWD [12] including mappings

We consider three multimedia applications as our case studies in this work: *MP3 and H263 encoder and decoder* (MMS) presented in [10], and *Video Object Plane Decoder* (VOPD) and *Multi-Window Displayer* (MWD) used in [12]. Figure 2, Figure 4.a, and b show the communication task graphs of these applications respectively.

Table 1 Spare Link Selections

Faulty Switch		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Alter Switches	MMS	2	5	7	3	1	10	4	11	6	13	14	8	9	15	16	12
	VOPD	5	1	7	3	9	10	2	4	6	11	12	8				
	WMD	2	1	4	8	6	3	10	12	5	9	7	11				

We greedily mapped these scheduled and bounded CTGs onto 4×4 and 3×4 mesh-based NoC architectures. These mappings are shown by numbers inside each core of Figure 2 and Figure 4.

Table 1 shows the alternative switch selection results for MMS, VOPD, and MWD applications.

6. Analysis and Results

This section shows how we analyze the proposed fault-tolerant architecture in terms of reliability and performance and present experimental results for three multimedia applications introduced earlier.

6.1. Reliability Evaluation Method

The reliability evaluation method that we present here is application dependent and based on the routing strategy used in the NoC. As discussed earlier, the CTG of an application mapped onto the NoC defines the communication behavior between the processing cores and determines the source and the destination of packets traveling through an NoC. Based on the routing strategy, for each pair of source and destination PEs there may exist one or more paths in the NoC each consisting of a number of switches. A packet traverses a path by passing through this set of intermediate switches.

The application mapped onto NoC cannot work properly unless all the packets reach to their destinations according to the behavior expected by the application's CTG. Therefore we can consider the NoC system as a set of paths between the cores so that failure of any of them causes the entire system to fail. So, like what is defined for the reliability of series systems [13], we define the reliability of NoC, R_{NoC}, as the product of reliabilities of all the paths $R_{i,j}$ between the cores in the CTG, as shown in Eq. 2. It is worth mentioning that since each path between two cores in the CTG is used only once, its reliability should not be raised to any power greater than 1. So in Eq. 2 we replace every occurrence of $R^k_{i,j}$ by $R_{i,j}$.

$$R_{NOC} = \prod_{(i,j)\in CTG} R_{i,j} \quad \text{Eq. 2}$$

$R_{i,j}$ for each pair of source and destination PEs, *(i,j)*, as demonstrated by Eq. 3, is defined as the probability of using an operational path for the communication between source i and destination j in the NoC. This is like what is defined as path reliability in point-to-point networks [14].

$$R_{i,j} = \sum_{k=1}^{N} p_k \prod_{m=1}^{S} R_m \quad \text{Eq. 3}$$

N is the number of paths between PE_i and PE_j and p_k is the probability of traversing k^{th} path for reaching to PE_j from PE_i. Since we just consider switch failures and assume that PEs do not fail, the reliability of k^{th} path between PE_i and PE_j can be defined as the probability of the proper working of all the switches traversed on this path. S is the number of switches and R_m is the reliability of m^{th} switch in path k.

6.2. NoC Reliability Analysis

In this section we calculate the reliability of the proposed NoC model before and after adding spare links. In the original NoC architecture of Figure 1 there is only one possible path between each pair of PEs in the NoC. This path consists of switches determined by *XY* routing algorithm. Therefore based on Eq. 2 the overall reliability of NoC is equal to the product of reliabilities of all the paths from sources to destinations. Therefore the final equation of NoC reliability equals to the product of reliabilities of all switches, as shown in Eq. 4. In this equation S is the total number of switches in the NoC and R_{SWi} is the reliability of SW_i.

$$R_{sys} = \prod_{i=1}^{S} R_{sw_i} \qquad \text{Eq.4}$$

After making the NoC architecture fault-tolerant a number of paths will be added to the normal paths taken by *XY* routing which should be considered in the NoC reliability formula. Referring again to the example of Figure 3, the paths between PE_{11} and PE_3 are shown in Figure 5.a. If SW7 fails (the black block), there would be two new alternative paths from the right or left each with determined probabilities of p_r and p_l. Note that these new paths will only be used when SW_7 fails; thus according to conditional probability, $(1\text{-}R_7)$ is multiplied to the reliability of them. Thus based on the Eq. 2 from the previous section, the reliability of the path between PE_{11} and PE_3 ($R_{11,3}$), considering only the possibility of failure in SW_7 will change to:

$$R_{11,3}=R_{11}[R_7+(1\text{-}R_7)\ (p_rR_{12}R_8R_4+p_lR_{10}R_6R_2)]R \qquad \text{Eq. 5}$$

Eq. 5 shows a reliability increase in the path between PE_{11} and PE_3 compared to the reliability before making the modifications to NoC, which was:

$$R_{11,3} = R_{11}\, R_7\, R_3 \qquad \text{Eq. 6}$$

Figure 5.b, c and d show the normal and alternative paths for three other cases of Figure 3. The reliability equation in each case is similar to Eq. 5.

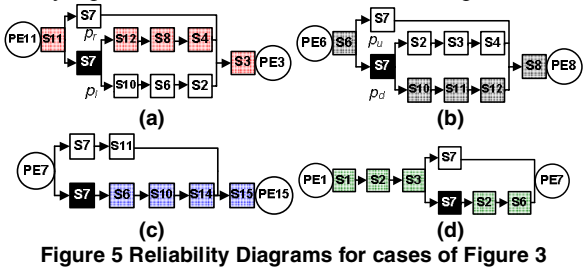


Figure 5 Reliability Diagrams for cases of Figure 3

The possibility of failure in each of NoC switches are considered in the final reliability equation of the fault-tolerant NoC. According to conditional probability, the reliability of each path in NoC is equal to the reliability of path in the case of no failure, in addition to the reliability of new re-routing paths used when one of switches in the path fails. The complete reliability equations for the examples of Figure 5, based on selected alternative switches of Table 2, are shown in Table 3.

Table 2 Spare link selection of Figure 3

Faulty SW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Alternate SW	5	1	2	3	9	10	6	4	13	11	7	8	14	15	16	12

Table 3 Reliability equations of paths of Figure 3

$R_{11,3}$	$R_{11}R_7R_3+(1\text{-}R_{11})R_7R_3 + (1\text{-}R_7)R_{11}\ (p_rR_{12}R_8R_4 +p_l\ R_{10}R_6R_2)R_3+(1\text{-}R_3)R_{11}R_7R_6R_2$
$R_{6,8}$	$R_6R_7R_8+(1\text{-}R_6)R_{10}R_{11}R_{12}R_8+(1\text{-}R_7)\ R_6\ (p_uR_2R_3R_4 +p_dR_{10}R_{11}R_{12})R_8+(1\text{-}R_8)R_6R_7R_3R_4$
$R_{7,15}$	$R_7\ R_{11}R_{15}+(1\text{-}R_7)R_6R_{10}R_{11}R_{15}+(1\text{-}R_{11})R_7(p_r\ R_8R_{12}R_{16} +p_lR_6R_{10}R_{14})R_{15}+(1\text{-}R_{15})R_7R_{11}R_{12}R_{16}$
$R_{1,7}$	$R_1R_2R_3R_7+(1\text{-}R_1)R_5R_6R_7+(1\text{-}R_2)R_1R_5R_6R_7 +(1\text{-}R_3)R_1R_2R_6R_7+(1\text{-}R_7)R_1R_2R_3R_6$

According to the above explanations the path reliability equation of Eq. 2 can be simplified, and based on equation Eq. 1 the reliability of our proposed fault-tolerant NoC architecture will be equal to:

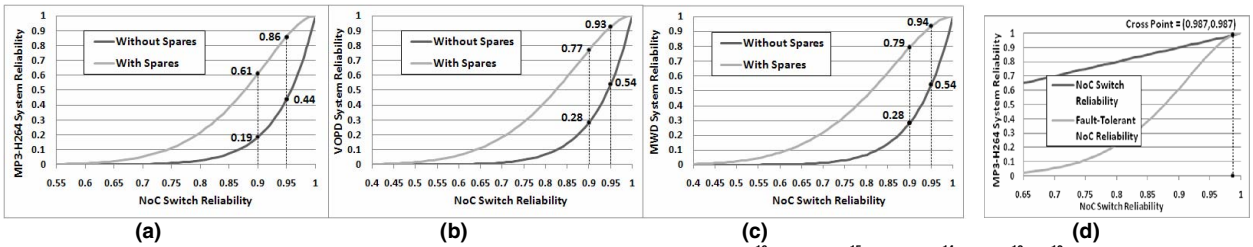
$$R_{NOC}=\prod_{(i,j)\in CTG}[\prod_{k=1}^{S} R_k+\sum_{k=1}^{S}(1-R_k)\cdot(R_{i,j}\mid SW_k\ Fails)] \qquad \text{Eq. 7}$$

7. Experimental Results

The reliability equations extracted for each of our three multimedia applications according to Eq.7 and their corresponding reliability curves based on different switch reliability values are shown in Figure 6.a, b, and c, respectively. In order to compare the reliability achieved by the proposed fault tolerant architecture, the results of the original NoC are also shown.

As simulation results show, the reliability of NoC for all the three experimented applications has been multiplied by about 3 and 2 when the reliability of the NoC switches is 0.9 and 0.95 respectively (Table 4). Furthermore, our fault-tolerant architecture leads to a system reliability of equal and even greater than the reliability of a single NoC switch for switch reliabilities greater than 0.96-0.98. As Figure 6.d shows, the reliability of NoC system for the MP3-H264 application crosses the NoC switch reliability at 0.987. This means that for NoC switches with reliabilities higher than 0.987 the overall reliability of the fault-tolerant NoC is more than the reliability of a single NoC switch used in the system. The cross points for VOPD and MWD applications are switch reliabilities of 0.969 and 0.964.

Furthermore, in order to have a performance estimation of the proposed reliable architecture, we use the TLM NoC simulator of Section 5 and estimate the overall average response time of system using Eq. 1. The overall performance in terms of average response time of system has decreased by just 2-3 percent for VOPD and MWD applications (Table 4). However, since directing packets through alternative switches sometimes leads to traveling in shorter paths from source to destination cores, we even see a performance improvement in the results for MP3-H264 application.



(a) (b) (c) (d)

Figure 6 (a) MP3/H263 System Reliability: $R(t) = 13.625R^{16} - 30.75R^{15} + 8.625R^{14} + 8.5R^{13} + R^{12}$
(b) VOPD System Reliability: $R(t) = 5R^{12} - 12.5R^{11} + 7R^{10} - 4.5R^{9} + 6R^{8}$
(c) MWD System Reliability: $R(t) = 2R^{12} - 9R^{11} + 19R^{10} - 27R^{9} + 16R^{8}$
(d) Cross Point of Switch and NoC Reliability for MMS System

Table 4 Reliability Improvement and Performance Cost for MP3-H264, VOPD and MWD

Application	Reliability Improvement		Performance Cost
	R_{SW}=0.9	R_{SW}=0.95	
MP3-H264	229.04%	95.51%	-3.36%
VOPD	172.52%	71.62%	2.8%
MWD	180.63%	73.13%	1.22%

8. Conclusions

To recover from permanent single faults in NoC switches, a fault-tolerant semi-regular NoC architecture has been proposed. In this architecture, a spare link is added between each processing element and one of its neighboring switches and a re-routing strategy is developed to bypass the failed switch. The best spare links to be added to this architecture are chosen by an architecture exploration algorithm, based on performance measurements achieved from high-level simulation of the target NoC. To estimate the achieved improvement in reliability, an application specific routing-based reliability analysis method is developed.

For our experimented applications, the analytical results show a reliability improvement of a factor of 2 to 3 for the entire NoC when the switch reliabilities are between 0.9 and 0.95. The overall performance of the architecture in terms of average response time of system has decreased by just 2-3 percent. The main overhead of the proposed fault-tolerant architecture is the probable increase in the power consumption and hardware due to the additional checks for detecting failed switches and re-routing packets.

The reliability analysis method presented here can be used in other NoC architectures with different topologies and routing strategies for evaluation of more complex systems.

9. References

[1] T. Bjerregaard and S. Mahadevan. "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no.1, pp.1-51, 2006.

[2] D. Park, C. A. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring Fault-Tolerant Network-on-Chip Architectures," in *Proceedings of Dependable Systems and Networks* , 2006, pp. 93-102.

[3] M. Ali, M. Welzl, S. Hessler, and S. Hellebrand, "An Efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip," *International Journal of High Performance Systems Architecture ,* vol. 1, no. 2, pp. 113-123, 2007.

[4] A. Shahabi, N. Honarmand, H. Sohofi and Z. Navabi, "Degradable mesh-based on-chip networks using programmable routing tables," *IEICE Electron. Express*, vol. 4, no. 10, pp.332-339, 2007.

[5] P. Rantala, T. Lehtonen, J. Isoaho, J. Plosila: "Fault-tolerant Routing Approach for Reconfigurable Networks-on-Chip," in *Proceedings of International Symposium on System-on-Chip,* 2006, pp.1-4.

[6] 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*, 2006, pp. 457–465.

[7] T. Lehtonen, P. Liljeberg and J. Plosila, "On-line Reconfigurable Self-Timed Links for Fault Tolerant NoC," *VLSI Design*, vol. 2007, Article ID 94676, 13 pages, 2007.

[8] U. Y. Ogras, and R. 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*, vol. 14, no.7, pp.693-706, July 2006.

[9] T. Simunic, K. Mihic, G. De Micheli, "Reliability and Power Management of Integrated Systems," in *Proceeding of Euromicro Symposium on Digital System Design,* 2004, pp.5-11.

[10] J. Hu, and R. Marculescu. "Energy-Aware Mapping for Tile-based NoC Architectures Under Performance Constraints," in *Proceeding of Asia & South Pacific Design Automation Conference*, 2003, pp. 233- 239.

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

[12] A. Jalabert, S. Murali, L. Benini, G. De Micheli, "xpipesCompiler: A Tool for Instantiating Application Specific Networks on Chips", in *Proceedings of the conference on Design, automation and test in Europe*, 2004, p.20884.

[13] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*, Morgan-Kaufman, 2007.

[14] S. Hariri and C. S. Raghavendra, "SYREL: A Symbolic Reliability Algorithm Based on Path and Cutset Methods," *IEEE Transactions on Computers,* Vol. C-36, pp. 1224–1232, October 1987.