Application Specific Configuration of a Fault-tolerant NoC Architecture

Fatemeh Refan[1], Parisa Kabiri[1], Homa Alemzadeh[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, E-mail :{ refan, pani, homa, navabi}@cad.ece.ut.ac.ir*
[2] *Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy, E-mail: Paolo.Prinetto@polito.it*

ABSTRACT: This paper discusses the configuration of a fault-tolerant mesh-based NoC architecture. In this architecture, spare links provide a mechanism for rerouting data packets in presence of NoC faults. Two algorithms, Exhaustive and Greedy, are used to find the best configuration. The NoC is modeled at the high transaction level using SystemC TLM. This enables easy design space exploration and performance analysis of the proposed NoC architecture. The performance results driven from simulation of this model are used as the input of the spare link selection algorithms.

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 with different NoC components [2]. This emerges the need for designing self reconfigurable methods, capable of recovering from faulty situations, without discarding the whole IC.

The most popular academic and industrial NoC architecture has mesh topology [3]. Several studies have been done to introduce self-healing techniques to recover from faults in this architecture. Since switches are the critical traffic points of mesh-based NoC, methods that focus on the switch faults are at the highest importance. Most of presented techniques reconfigure the system by updating the routing tables [4], [5], [6], and/or add extra information to the packets aiming at fault tolerant routing strategy, in addition to adding hardware and spare insertion [7], [8].

In [9] we have combined both of above methods to recover from faults in switches. We prevented both problems that occur when a switch fails: inaccessibility of the switches directly connected processing element (PE) to/from network, and unusability of the failed switch for directing the incoming packets to their destinations.

The fault-tolerant NoC architecture proposed in [9] recovers from switch failures by adding a redundant link between each PE and one of its neighboring switches, and making modifications to the NoC components, in addition to use a rerouting strategy. Continuing this work, in this paper we explain the simulation method used for finding the best spare links to configure the NoC. Then we use two algorithms, Exhaustive and greedy, to accomplish the above goal.

The rest of this paper is organized as follows. In Section 2 we describe the simulation method. Section 3 presents spare link selection algorithms. We report the results obtained by simulation and algorithms in Section 4. Finally we conclude in Section 5.

2 Simulation Method

The fault-tolerant NoC architecture of [9] has been implemented in SystemC TLM 2.0 library [10]. Switches and PEs 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 used to model *blocking* and *nonblocking* unidirectional data transfers within our NoC model. We take advantage of TLM approximate-time abstraction level for modeling and performance evaluation of the NoC system. Therefore, without considering timing details and just by using a number of *wait()* statements we can model and simulate the traffic seen by each packet in every switch at a high-level.

Furthermore, since we concentrate on the switches as communication elements of NoC and consider spare links for recovery from faults, the critical bottleneck becomes the communication performance. Therefore we just focus on the communication behavior of NoC architecture and ignore the computation details of processing elements in our model. TLM features for separation of communication and computation parts within a system help us in reaching this goal. We describe the processing elements by just a rough high-level modeling of their timings, implementing their data communication and ignoring the precise functionalities.

At the start of simulation, each PE distinguishes its own tasks and stores the information about its incoming and outgoing packets from a specific input application defined by a communication task graph (CTG). 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.

978-1-4244-2060-5/08/$25.00 ©2008 IEEE

We consider the worst case response time of the sink PEs and the average response time of the system as the evaluation parameters for our proposed architecture. When a packet arrives at its destination, its delay is calculated based on the timing information that it carries. This timing information includes the number of passed switches, inter-switch links and network interface to switch wrappers, PEs observed, and the turns wasted waiting in the input FIFOs of the 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 the delay value which represents relative values is sufficient for comparison of the results. So the delay for each packet is calculated as shown in Eq.1. Consequently, the average response time of the system is estimated by the maximum delays of last packets arriving at the sink PEs.

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

3 Spare Link Selection

After Performing above simulation, we need to choose only one spare link for each PE. To solve this problem we use Exhaustive and Greedy search algorithms to explore the space of different spare link selection possibilities for the PEs of an NoC. We choose the best alternative switches for all PEs for the input application based on the average response time of the system in the case of selecting each set of alternatives.

First we present inputs, outputs, and constraints of spare link selection problem. We get the final timing results driven from simulation as a two dimensional input array *T*. On the other hand, output is a one dimensional array *A*, representing the alternative switches; the precise definition of arrays *A* and *T* is shown in Eq.2.

$$T_{i,j} = \begin{cases} +\infty, & SW_j \text{ is not neighbour of } PE_i \\ \text{Delay of choosing } SW_j \text{ as spare of } PE_i, & o.w. \end{cases}$$

$$A_i = \text{the ID of spare switch of } PE_i \qquad \text{Eq.2}$$

To define the constraints, first we should introduce some important terms; blank PE slot, neighbourhood of a switch and a PE, and finally plain layout. *Blank PE slot;* Considering a network of n×m switches, there are potentially *(n+1)* × *(m+1)* options for placing PEs, we call each of them a blank PE slot. Figure 1(a) shows all blank PE slots represented by empty circles in a 2×3 mesh-based NoC. *Plain layout;* The placement of PEs in blank PE slots is called plain, if no two PEs are placed in the same blank slot. E.g. the regular mesh-based architecture has a plain layout (Figure 1(b)). *Neighbourhood of a switch and a PE;* In mesh-based architecture, each PE is placed in the neighbourhood of its local switch. E.g. in the architecture of Figure 1(b), PEs are placed in right-up neighbourhood of switches; however, some switch can have more than one neighbouring PEs in its other directions (right-down, left-up, and left-down). E.g. PEs 1, 2, 5, and 6 are in the neighbourhood of switch 1 in Figure 1(b).

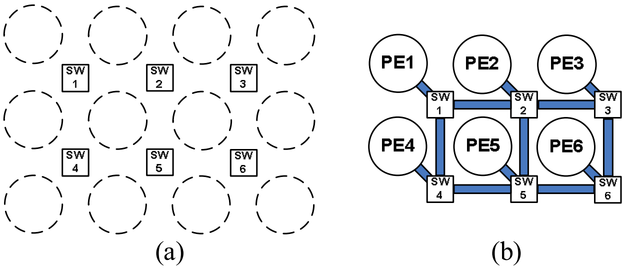


Figure 1 2x3 NoC Mesh-based NoC
(a) Blank PE slots, (b) NoC Architecture

Using above definition, this problem has two constraints:

1. As stated before, each switch is limited to having only one spare link connected to it; i.e.
$$\forall i,j: 1 \le i,j \le n \times m: \ A_i = A_j \ \Rightarrow i = j$$
2. In order to have a plain layout, there should be one placement offered by array A such that:
   a. Each PE is placed in the neighborhood of its local and spare switches
   b. No two PEs are placed in the same blank slot

The Exhaustive algorithm tries all valid placements satisfying constraint 2, and eliminates the options among them which do not satisfy constraint 1. To satisfy above considerations, we introduce *O* function which is a two dimensional array, representing the placement of a PE; i.e. for $\forall i,j: 0 \le i \le n, 0 \le j \le m$, Eq.3 should hold.

$$O_{i,j} = \begin{cases} -1, (i,j) \in \{(0,0),(0,n),(m,0),(n,m)\} \\ k, PE_k \text{ is placed in slot } (i,j); k \in PP_{i,j} \\ 0, \text{no } PE \text{ is placed in slot } (i,j) \end{cases} \qquad \text{Eq.3}$$

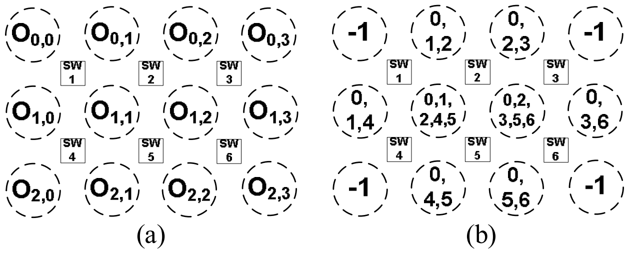


Figure 2 O function:
(a) Representing placement, (b) The function value(s)

For invalid blank PE slots, $O_{i,j}$ is -1, otherwise it is equal to the ID of the PE which is placed in empty slot *(i,j)*. In the case that no PE is placed in this slot, its value will be 0. The invalid cases are corner blank PE slots. The reason is that in the fault tolerant architecture, each PE is attached to two switches. Therefore no PE can be placed in the four corners. The way that *O* function represents blank PE slots are demonstrated in Figure 2(a). Furthermore, only the PE of neighboring switches of each blank

PE slot can be placed on it. The set of possible IDs for each slot is shown in Eq.4, and shown for a 2×3 NoC in Figure 2(b).

$$PP_{i,j} = \{(i-1)m + j, (i-1)m + j + 1, im + j, im + j + 1, \\ \mid i - 1 \geq 0 \; and \; j - 1 \geq 0\} \qquad \text{Eq.4}$$

In the Greedy algorithm, we try to find a result that seems to be best in the moment. In each step we minimize the multiplication of communication volume and delay for each switch. We start by sorting our switches descending, based on their communication volume. Furthermore for each switch, we sort its possible spare switches descending, based on the delay of the two switches. Starting with the switch with highest communication volume, for each switch, the neighbouring switch with lowest delay that is not previously chosen as spare of another switch is selected. In each step, we check both constraints. In this method, although for each switch the optimal choice is selected, this necessarily does not lead to the optimum result for the whole system.

```
1  for i from 1 to nxm
   {
2    for j from 1 to switch[i].count
     {
3      if (!selected[switch[i]
             .connectedTo[j].toSwitch])
       {
5        selected[switch[i].connectedTo[j]
             .toSwitch]= true;
6        switch[i].selected = j;
7        ans+=switch[i];
             .connectedTo[j].toMinimize;
8        if (checkConstraints(i, j))
9          break;
       }
     }
10   if (!switch[i].selected)
     {
11     switch[i-1].selected = 0;
12     i--;
13     GoTo 2;
     }
   }
14 answer = ans;
```

Figure 3 Pseudo code of Greedy algorithm

Figure 3 shows pseudo code of Greedy algorithm. The algorithm, first checks whether switch[j] is spare of switch[i] if not, chooses it as the spare and sets this choice for both switches. Then in *checkConstraints* function, constraints are checked: If the result satisfies them, it continues to next switches. Otherwise, the selection is ignored and another spare is chosen for switch[i]. Finally, if the algorithm can select no switch as the spare for a switch, it goes back to the previous switch and changes its choice. This continues until a feasible solution is found.

n×*m* is size of NoC. *Selected*, is an integer variable that shows which switch is selected as result. *ConnectedTo* is an integer value that saves the spare switch for each switch. *Tominimize* is a float variable, representing communication volume multiplied by delay for each switch.

4 Experimental Results

We consider three multimedia applications as case studies in this work: *MP3 and H263 encoder and decoder* (MMS), *Video Object Plane Decoder* (VOPD) and *Multi-Window Displayer* (MWD) [9]. Figure 4, Figure 5(a), and Figure 5(b) show the CTG of above applications mapped onto the cores, respectively.

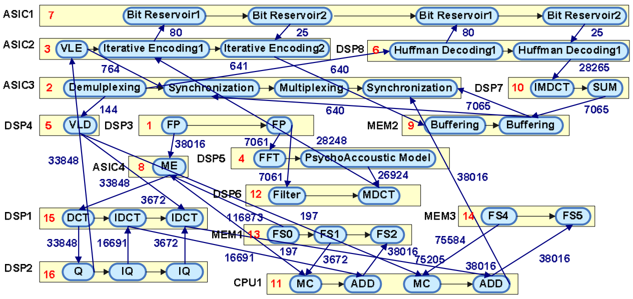
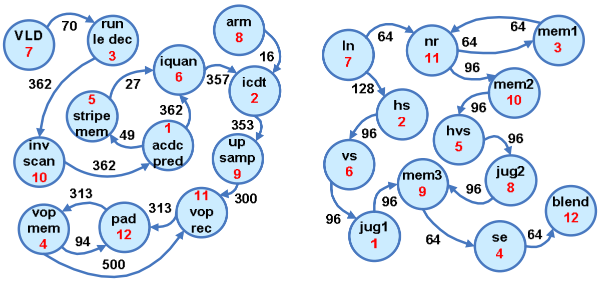


Figure 4 CTG of a Complex Multimedia Application (MMS) [9]



(a) VOPD (b) MWD

Figure 5 CTGs of VOPD and MWD [9] including mappings

Figure 6 shows the average response time of MMS system in the case of choosing each of the possible alternative switches. The first column shows the response time in the presence of no failure and the next columns show the results for switches SW_8, SW_{11} and SW_{13} as three cases with different locations in the NoC and different number of neighboring switches to be selected as possible alternatives. It is clear from the results that the best alternative switches to be chosen for SW_8, SW_{11} and SW_{13} are SW_{11}, SW_{14}, and SW_{10} respectively. The simulation results prove the analytical estimations. For example *ASIC4* core which is mapped onto tile 8, has direct communication with tiles 1, 11, 13, and 15. This suggests that if the spare link is chosen to be connected to SW_3 or SW_4, the average number of switches that *ASIC 4*'s outgoing and incoming packets should pass increases, while choosing the other three choices, SW_7, SW_{11}, and SW_{12}, can improve the overall performance by decreasing the distance between source and destination tiles.

Table 1 shows the alternative switch selection results of Exhaustive and Greedy algorithms for MMS, VOPD, and MWD applications which are mapped onto 4×4, 3×4, and 3×4 NoCs, respectively. The plain layout achieved by Exhaustive algorithm for MMS application CTG is shown in Figure 7, where the lighter connections between PEs and switches show spare links.

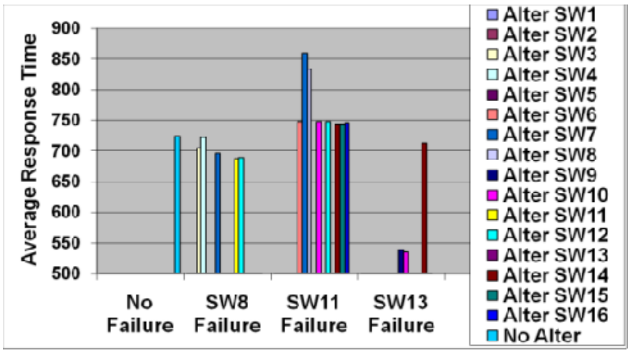


Figure 6 Alternative Selection Results for Switches 8, 11, 13

Table 1 Spare Link Selections

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

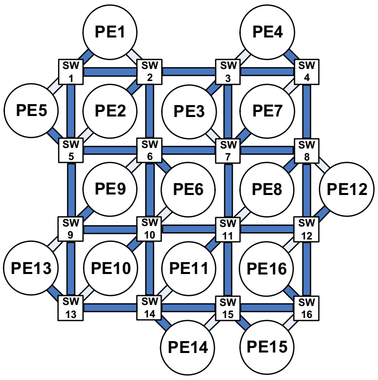


Figure 7 Possible layout of Exhaustive search results for MMS

Table 2 represents the average response time of each application for both methods. As it is clear, the average of delays in Exhaustive algorithm is less than Greedy results. Thus, Exhaustive algorithm is better than Greedy algorithm in terms of finding optimum solution, but the cost that we should pay for this is the higher time order of this algorithm.

Table 2 performance results for each application/algorithm

	Exhaustive	Greedy
MMS	704.23	704.97
MWD	764.75	768.75
VOPD	1,013.58	1,085.75

5 Conclusions

The work in [9] was continued by explaining the simulation method and proposing Exhaustive and Greedy spare link selection algorithms based on performance measurements performed by high-level simulations.

The presented TLM-based simulation method is also useful for easy modeling and fast exploration of NoC architectures. HW/SW co-design capabilities of TLM help simulating the execution of a specific input application mapped onto NoC and analyzing its performance factors, as fast as a software program. Moreover, with minor changes to the re-routing strategy, our configurable transaction level models for switches and PEs can be exploited in performance analysis of other NoC topologies as well.

Spare links selected by Exhaustive algorithm offer the architecture with the highest performance; however the order of algorithm is much higher than the Greedy algorithm. This tradeoff emerges use of Greedy algorithm for NoCs of greater size.

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] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson,J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits,* vol. 43, no. 1, pp. 29-41, 2008.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] F. Refan, H. Alemzadeh, S. Safari, P. Prinetto and Z. Navabi, "Reliability in Application Specific Mesh-Based NoC Architecture," to be published in *proceedings of 14th IEEE International On-Line Testing Symposium, 2008.*

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