Programmable Routing Tables for Degradable Mesh-Based Networks on Chips

A. Shahabi, N. Honarmand and Z. Navabi
CAD Laboratory, School of Electrical and Computer Engineering, University of Tehran, Tehran, IRAN
{shahabi, nima}@cad.ece.ut.ac.ir, navabi@ece.neu.edu

Abstract: *The decreasing manufacturing yield of integrated circuits, as a result of rising complexity and decreased feature size, and the emergence of NoC-based design techniques, has necessitated the search for network reconfiguration techniques for reusing NoCs with faulty communication hardware. In this paper, we propose a method to cope with the problem of faulty ports in NoCs with mesh topology. The method is based on the use of reconfigurable routing tables in network switches. We investigate this technique for a conventional routing mechanism and our optimized routing approach. The conventional mechanism uses one entry in its routing table for every destination address while our proposed routing mechanism uses a fixed number of entries per table and routes based on the address value comparison of the current switch and the destination switch. Experimental results show that a network reconfigured for fault masking by programming its routing tables has acceptable but degraded performance parameters as compared to the original, non-faulty network.*

Keywords: Degradability, Network on Chip, Programmable Routing Table.

1. Introduction

By the end of the decade, 45-nm transistors operating below 0.7 volt will be used which make it possible to put 4 billion transistors running at 15 GHz on a single chip [1]. System-on-chip (SoC) design methodology [2] has been proposed to provide the appropriate and integrated solutions to manage the increased complexity inherent in these large chips. In this technique the pre-designed and pre-verified block, called Intellectual Property (IP) core, are achieved from the third parties or internal source and combined on a single chip to facilitate the design process and reduce time to market. Communication between the IP cores is one of the most important challenges in the SoC design methodology. Bus-based architectures, which are generally used in the common SoCs, cannot closely follow the process evolution.

Network on Chips (NoCs) [3], [4] have been proposed to solve this problem by decoupling of the processing elements (*i.e.,* IP cores) from the communication fabric. A typical NoC at least consists of four major components: Processing Elements (PEs), Switches, Network Interface Units (NIUs), and Physical Links. PEs do the actual processing while the others constitute the communication fabric. NIUs are the interfaces of PEs to the network. Wormhole routing [5] is commonly used in the NoC architecture as the switching scheme. Several NoC architectures and their implementation details are presented in [6], [7], [8], [9] and [10].

While advances in silicon integration technology and nano-scale feature size have made it possible to put more and more transistors on a single die, it has also caused the dies to be more susceptible to manufacturing faults and decreased the process yield. Since manufacturing faults tend to be local and affect only a limited area of the IC, it is possible (and very desirable) to find methods to make such faulty ICs reusable. Such failures can occur in PEs and/or in communication fabric. Several methods have addressed this problem for the processing elements [11] and communication fabric [12]. In this paper, we focus on alternative solutions for recovering a faulty communication fabric.

One possible method to work around the permanent faults is to use fault tolerant architectures. Marculescu [13] has discussed the possibility of achieving on-chip fault-tolerant communication based on a randomized gossip protocol. A Node forwards the packets to a randomly chosen subset of the neighboring nodes until the packets reach the destination. Detection and handling of the duplicate packets in each node increase the complexity and overhead of routing.

Yang et al. [12] have proposed a new torus [7]-like network and a fault-tolerant routing algorithm to compensate the effect of link failure(s). In that work, the basic torus is called rank-0 torus. Each next rank is formed by adding four links (at an angle of 45 degrees) to the previous one. Obviously, the added links impose a considerable area overhead on the design and increase its wiring complexity.

Reconfiguring the faulty IC to avoid using the faulty modules (PEs, switches or links) and get the work done using the remaining non-faulty ones is another approach, that is usually referred to as Degradability [11]. Traditionally, reconfiguration based techniques have been used for highly regular circuits (*e.g.* memory chips), but in the realm of NoC design methodology, effective methods should still be sought for.

Generally, degradability-based methods imply a design and manufacturing flow including the following steps: First, at the design stage, special algorithms

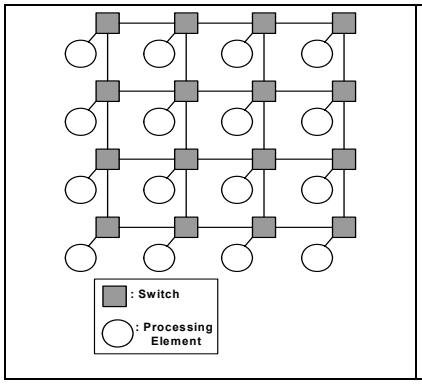


Figure 1: Mesh Topology

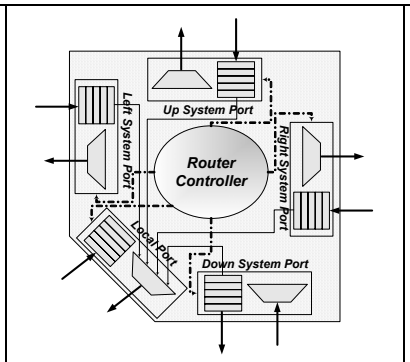


Figure 2: Simple switch architecture for the mesh topology

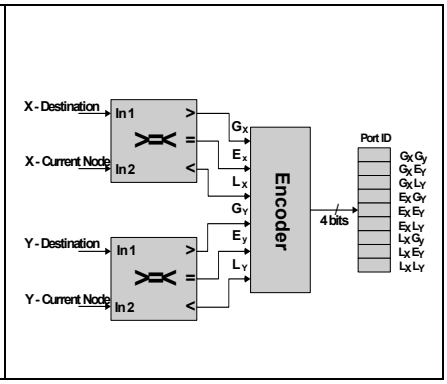


Figure 3: Mesh-based routing mechanism

should be used which result in reconfigurable data paths and control units. Then, after the IC has been built, diagnosis techniques [14] should be used to detect the faulty components of the circuit. Then, based on the obtained fault pattern, a proper configuration, which bypasses the faulty modules, should be chosen and programmed into the circuit. This implies that the circuit should inevitably have some programmable elements in it and post-manufacturing reconfiguration should be possible. But, this is by no means a great overhead because widely-used built-in provisions such as JTAG or scan chains can be used for this purpose.

In this paper, the problem of faulty ports in the switches is considered for a network with mesh topology. In Section 2 we propose a new routing mechanism for mesh-based networks instead of the conventional routing mechanism which has serious drawbacks. Then, in Section 3.1 and 3.2, we provide algorithms to reconfigure a network with faulty ports that use the conventional and proposed routing mechanism respectively. Finally, in Section 4, we compare the performance parameters of the reconfigured networks with faulty links against those of original non-faulty network.

2. Switches and Routing Mechanisms

Kumar et al. [6] have proposed a simple mesh-based interconnection architecture, called CLICHÉ (Chip Level Integration of Communication Heterogeneous Element), as an NoC architecture (Figure 1). In this topology, every switch, except those placed on edges, is connected to four adjacent switches and to the related PE. The switches at the edge are connected to two or three neighboring switches. The mesh-structure is commonly used in the NoC architectures because of its simple layout structure and independency of local interconnections from the size of the network.

In an NoC, switches are responsible for routing the packets between nodes. Each switch has a set of bidirectional ports through which it is connected to neighboring switches or PEs. It also contains a router to define a path between input and output ports, buffers to store intermediate data and an arbiter to grant access to a given port when multiple input requests arrive in parallel. Simple switch architecture for the mesh topology is shown in Figure 2. A mesh switch, in general, has five bidirectional ports: one attached to the local PE (local port) and the other four to the neighboring switches (system ports). Each port consists of a FIFO and a multiplexer. The FIFO Buffers the incoming packets and the output selection is done through the multiplexer. If the received packet destined for that node, it should be delivered to the attached PE, through the local port. Otherwise, one of the output system ports (Left, Right, Up and Down) should be selected according to the destination address.

Implementation of the routing algorithm is one important aspect of the switch design process. It can be implemented as a hardwired module or as a Programmable Routing Table (PRT). In the hardwired style, no programming is required but the switch cannot be altered to bypass the faulty port(s). Thus, the chip will become unusable whenever some ports become faulty unless fault-tolerance provisions have been made into the circuit. But, in the case of PRTs, the switch can be programmed to bypass the invalid ports and get the job done using the remaining non-faulty ports. As said before, this approach, inevitably, adds an additional step of programming the PRTs to the manufacturing process.

The paths taken by packets between source and destination switches, hence the contents of the PRTs, are defined by the routing algorithm in use. A routing algorithm should be able to prevent live-lock situations [5]. Live-lock refers to the situation in which some packets will not reach their destination, even if they never get blocked permanently.

The simplest way to implement an PRT is to use a lookup table with as many entries as the number of nodes in the network. The index of the table will be the destination address of a packet and each entry will contain the identifier of the proper output port for the given destination address. We call this method Per-Address Routing (PAR) because the PRT has one entry for each possible address in the network.

The PAR-based PRT, in spite of its simplicity, suffers from two major drawbacks: (1) The size of the lookup table will grow linearly with the number of NoC nodes. In addition to the area overhead, the switch

throughput will decrease due to the large size of the table and the resulting delay of lookup operation (table lookup should take place at least once for every received packet). (2) The PAR method is not amenable to network scaling because the switch cannot be used in networks with more nodes than the number of lookup table entries.

Based on the particular topology and routing algorithm in use, there may be some methods to work around the above problems. We demonstrate such a technique for mesh topology and its widely used XY [5] routing algorithm.

In mesh-based networks, the address of a node is a pair (x,y) which is the coordinates of the node in the mesh. A simple live-lock free routing algorithm, called XY, has been proposed for the mesh topology. In this algorithm, a packet is first routed in X direction: if the x-coordinate of the destination is less than that of the current node it will be routed to the left and if it is more than that of the current node it will be routed to right. Afterwards, the packet will be routed in the Y direction, *i.e.*, up or down, based on the y-coordinate of the destination address.

Based on this algorithm, Figure 3 shows the proposed routing mechanism. Two small comparators compare the x- and y-coordinates of the current node with those of the destination. The outputs of each comparator can assume three different one-hot coded states (*G* for greater, *E* for equal and *L* for less). Thus, we can have 9 different situations for the combination of two comparator outputs. An encoder will generate a 4-bit signal to indicate the occurrence of one of these 9 situations. The output of the encoder is used to index a 9-entry, programmable lookup table. Each entry of the lookup table contains a port identifier to indicate one of the five ports that should be used. Obviously, the routing hardware has a fixed structure and does not depend on the number of nodes in the network. Throughout this paper, we refer to this method as Mesh-Based Routing (MBR) because it is tailored to the mesh topologies.

3. Programming PRTs Under Port Failures

When an output (input) port becomes faulty, the related switch cannot send (receive) any packet to (from) that port. Thus another non-faulty port in the switch should perform the task of the faulty switch. Each faulty port in a switch will make a non-faulty port in the adjacent switch unusable. For example when the left input port of switch B (Figure 4(a)) goes faulty, the right output port of the adjacent switch (A in Figure 4(a)) will become unusable.

The network topology can be considered as a directed graph with switches being its vertices and links being its edges. If this graph is strongly connected, then the network will be **structurally connected**. We say that the network is **routing-connected** if for every source and destination (*SRC*, *DST*) pair of nodes, a packet produced in *SRC* can be routed to reach *DST*. Whether this is possible or not depends on the routing table configuration of the nodes that the packet visits. Improper configurations might cause a live-lock and prevent the packet from reaching its destination.

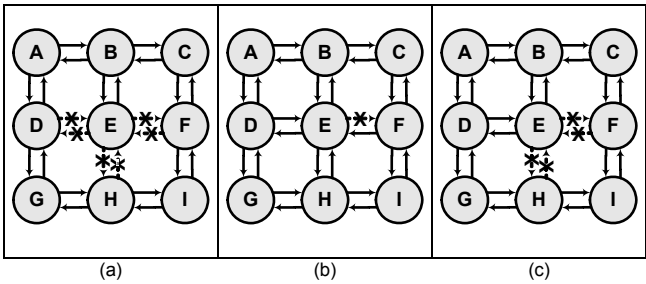
It is possible for a network to be structurally connected and yet no set of PRT configurations can be found to make the network routing-connected. We call a network **routing-connectable** if there is a set of PRT configurations to make the network routing-connected.

In Section 3.1, we show that every structurally-connected network with PAR-based PRTs is routing-connectable. In Section 3.2 we show that a structurally-connected network with MBR-based PRTs might not be routing-connectable, and we provide an algorithm that tries to find a proper set of PRT configurations if the network is routing-connectable.

3.1 Programming PAR-based PRTs

It can be shown that if a network is structurally connected, then there is a set of PAR-based PRTs for the network to become routing-connected. Such PRT configurations can be found using any live-lock free routing algorithm such as the shortest path. To do this, we first find the shortest path between all nodes. Then, Consider a particular (*SRC*, *DST*) pair of nodes. Suppose that, in the shortest path between *SRC* and *DST*, N_{SRC} is the node appearing after *SRC* and *SRC* is connected to N_{SRC} through port P_N. Then, in the PRT of *SRC*, set the entry for *DST* to P_N. If we repeat this for all possible node pairs, the resulting PRT configurations will shape a routing-connected network.

This algorithm proves that every structurally connected network with PAR-based routing tables is routing-connectable. However, the obtained network might not have good performance parameters, e.g., path length and network congestion.



Switch	Hard Constraints	Soft Constraints
B	D < R	D < L
C	D < L	-
D	R < D	R < U
E	-	R < U, D < L
F	-	L < U, L < D
G	R < U	-
H	-	U < L, U < R

(d)

Figure 4: (a) a network which is not routing connectable using MBR-based routing tables (b) a network with one faulty port (c) a network with four faulty ports (d) routing constraints in network of (c)

```
CONFIGURENETWORK(failure_set, current_constraints)
begin
  if ISEMPTY(failure_set) then
    for each configuration conf which satisfies all the constraints
      test all possible (src, dst) node pairs for a live-lock free path
      if the test succeeds then
        return conf;
      end if;
    end for;
    return NULL;
  else
    f = FIRST(failuer_set);
    n = node affected by f;
    for each possible helping node h for n
      new_constraints =
        current_constraints +
          {constraints from using h as the helper};
      if COULDBESATISFIED(new_constraints) then
        return
          CONFIGURENETWORK(failure_set - f, new_constraints);
      else
        return NULL;
      end if;
    end for;
  end if;
end
```

Figure 5: Reconfiguration algorithm for MBR-based routing tables

3.2 Programming MBR-based PRTs

It is possible for a MBR-based network to be structurally connected but not routing-connectable. Figure 4(a) shows an example of such a network. In this figure, the crosses indicate the faulty ports. For example cross between node D and node E indicates the right output port of node D or left input port of node E or both of them is faulty. No configuration can be found for the MBR-based PRTs to create a routing-connected network. Because when B has a packet destined for E, it should send it using BE link and when it has a packet destined for H it should not use BE link. Thus, there is no feasible port identifier for (E_x G_y) entry of B's routing table.

When a packet arrives at a switch, either it is destined for the attached PE or it must be routed using one of the four system ports. In what follows, we use L, R, D and U to indicate moving in left, right, down or up directions respectively. The decision on where to route the packet is based on the result of address comparison between current switch and the destination switch.

Consider a packet which is currently at a switch addressed (x_{cur}, y_{cur}) and is destined for switch (x_{dst}, y_{dst}). We define the distance of the switches as

$$dist(cur, dst) = (| x_{dst} - x_{cur} | + | y_{dst} - y_{cur} |)$$

This distance is the minimum number of hops that the packet should traverse until it reaches its destination. If a routing algorithm routes a packet in such a way that its distance from the destination switch decreases with each move, the algorithm will be live-lock free. This is because a live-locked packet will visit a switch twice and this cannot happen with a decreasing sequence of distances. The conventional XY routing algorithm has this property and thus is live-lock free. We call every move that decreases the distance of packet from its destination a **positive** move. Otherwise, we call it a **negative** move. In mesh-based networks, such moves will inevitably increase the distance.

Sometimes there are multiple positive moves for a packet. For example, if the destination is both to left and above the current switch, taking either direction will be a positive move. In this case, direction to choose depends on the routing table of the current switch. An MBR-based routing table has one entry for every possible combination of positive moves, as shown in Figure 3. The contents of this table indicate the relative priority of moving in different directions. For example, if in the entry corresponding to positive moves in up and left directions, (L_x L_y) in Figure 3, the port identifier of the left port is given, then the priority of moving in the left direction is more than moving up. We use the "<" operator to show the priority of movements. In this case, we write U<L.

Reconfiguration Procedure. When a port in a switch becomes faulty, the switch will need some help from one of the neighboring switches (helping switch) to route the packet. Since the switch with a faulty port might be forced to perform a negative move, the distance may increase and if the priorities in the helping switch are not set properly, it might return the packet to the original switch and cause live-lock. Hence, it will be necessary to put some constraints on the possible priority combinations that the helping switch can use. For each switch, we can show the relative priorities of different directions with a permutation of L, R, D and U. For example, with LUDR pattern (the leftmost letter has the lowest priority), the switch first routes the packets to right if it is a positive move. Otherwise, the switch considers the down direction. Finally, the left port is selected when no movement is available for the right, down and up port. The permutation of 4 elements can assume 24 different patterns. However, since the switch cannot use its Left (Up) and Right (Down) ports simultaneously, some of the patterns can be combined. The UDLR, UDRL, DULR and DURL (LRUD, LRDU, RLUD and RLDU) can be merged because they behave like familiar XY (YX) routing algorithm. Thus, only 18 different patterns can occur and should be considered in the algorithm.

An example with one faulty port. To demonstrate our technique, Figure 4(b) shows a mesh structure with one faulty port. Since the right output port in node E is faulty, switch E can select one of its three adjacent switches (B, D, and H) as the helping switch. Suppose that we choose switch B. This will impose some restrictions on the routing table of B to prevent live-lock situations. Suppose node E has a packet destined for node F. Since the right output port of E is faulty, node E will send the packet to B, instead. Now, node B has a packet that should move both right and down. If,

in routing table of B, down moves have a higher priority than right moves, B will send the packet back to E and will cause a live-lock. Thus, for B, the priority of down move should be less than right move, or D<R. This is a Hard Constraint (HC) for B, *i.e.*, it must be met to have a live-lock free routing.

When we consider all the possible faulty ports and extract all the required constraints, it might be the case that for a switch, the constraints are conflicting. For example, a switch might have both L<R and R<L constraints. Obviously, these are conflicting constraints and cannot be satisfied simultaneously. Suppose that in our example, we first choose B as the helping switch of E and after considering other switches, we get trapped in a conflicting situation. It might be the case that this conflict has happened because of choosing B as the helping node. Hence, we should now check the alternative choice and test the selection of H or D instead of B as the helping node. This means that our algorithm should have a backtracking nature and whenever it encounters some conflict, it should backtrack to the last point that it had an alternative choice and test that one.

Moreover, we can use another constraint type, called Soft Constraint (SC) to reduce the average length of paths in the mesh structure under faulty conditions. Suppose that in Figure 4(b), E is helped by switch B to route in the right direction. It is better to force switch E to first route the packets using its non-faulty links (up, left and down) if it is a positive move. This means that switch E will use the helping switch B to route when no other choice exists. Thus we consider the following SCs for switch E: {R<U, R<D, R<L}. Since, moving to right and left are exclusive and cannot happen at the same time, the (R<L) constraint is useless. Thus, we can add (R<U) and (R<D) to the soft constraint list of node E.

As another example, Figure 4(c) shows a network with four faulty ports (right and down output port of E, left output port of F and up output port of H) and Figure 4(d) shows the assigned hard and soft constraints for each switch. The switches missing in the table have no constraints.

Reconfiguration Algorithm. Figure 5 shows the reconfiguration algorithm. The CONFIGURENETWORK routine has two inputs: set of faulty ports that have not been handled yet, and the constraints resulting from handling previous faults. It selects one faulty port from the list and, one by one, examines all possible helping nodes for the node affected by the faulty port. It repeats this process until there remains no faulty port in the list, *i.e.,* all the faults get handled, or at some point, the set of constraints could not be satisfied. In the former case, it checks all the routing table configurations that satisfy the constraints. If a configuration results in a live-lock free network, it will be returned as the result. Otherwise, the algorithm will backtrack to the point of the last choice. In the latter case, the algorithm will consider alternative helping nodes for the node affected by current faulty port. If there is no unchecked alternative, the search fails and the algorithm will return NULL to indicate failure.

4. Experimental Results

To assess the proposed technique, we have considered 10 examples 4 by 4 networks, Figure 6, with 1 to 36 faulty ports. For example EX1 in this figure shows that the down output port of node (2,2) or the up input port of node (2,3) or both of them can be faulty. To measure the quality of obtained network configurations, we have extracted the following parameters for each network: the average and maximum path length in the network and the average and maximum link load. Load of a particular link indicates the number of different (*SRC*, *DST*) pair of nodes whose packets should use that link. Table 1 presents the results obtained for MBR-based techniques. The row indicated with XY in the table gives the performance parameters for a network with no faulty ports in which all the routing tables are configured according to XY routing mechanism. The values in parenthesis are normalized with respect to the corresponding values of non-faulty XY network.

As Table 1 shows, the general trend of performance parameters is to get worse when the number of faulty ports increases. However, there are some deviations. These deviations can be attributed to the sensitivity of the results to both the number and the pattern of faulty

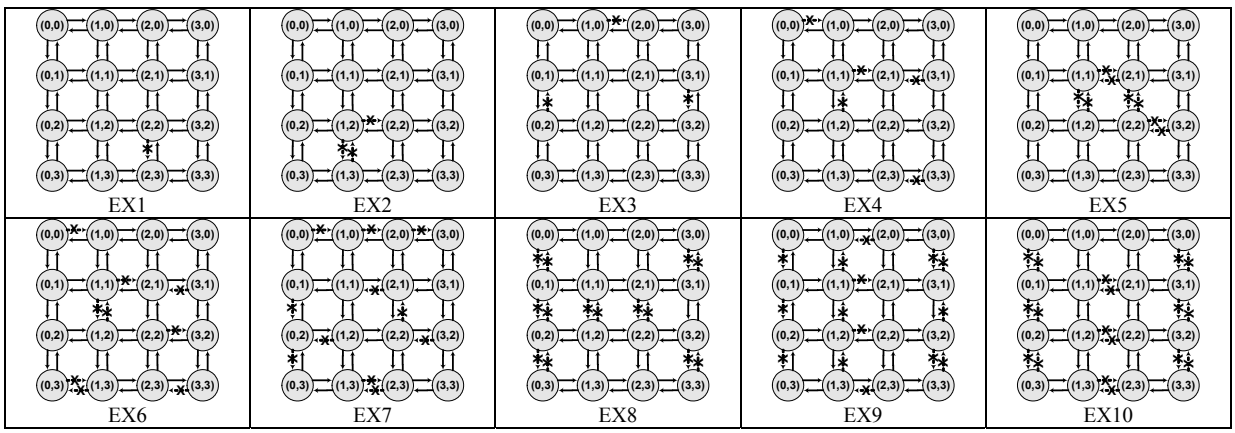


Figure 6: Example networks with 1 to 36 faulty ports used in experiments

ports in the network. EX2 and EX3 in Figure 6 demonstrate the sensitivity to pattern of faulty ports. In both cases there are 3 faulty ports and only the positions of the ports differ in two cases. As expected, when the algorithm is able to incorporate soft constraints, the results will be better than when only hard constraints are considered.

Also it can be seen that, although the maximum link load increases drastically as compared to non-faulty network, but other performance parameters, especially average values, increase smoothly with the number of faulty ports. It is interesting that the maximum and average path length increase by 50% and 42% respectively in the case of 36 faulty ports.

5. Conclusions

In this paper, we have considered the problem of faulty ports in NoCs and have proposed a method for reconfiguring the switch of a network to bypass the faulty ports. This method is based on using network switches with programmable routing tables. The reconfigured network may have a lower performance due to decreased number of available ports.

We have considered two different routing mechanisms to which we have applied the proposed technique. First, we showed that for every structurally connected network with PAR-based routing tables, a set of routing table configurations could be found to make the network routing connected. Also, we considered a new optimized routing mechanism and proposed an algorithm to reconfigure the networks using that routing mechanism. The experimental results show that the algorithm could reconfigure the faulty networks to achieve acceptable performance parameters with regard to non-faulty networks using XY routing mechanism.

Acknowledgment

The authors would like to thank the Iran Telecommunication Research center (ITRC) for supporting this work.

References

[1] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, World Semiconductor Council, Edition 2005, 2005.

[2] R. Saleh et al., "System-on-Chip: Reuse and Integration," *Proc. IEEE*, vol. 94, no. 6, pp 1050-1069, Jun 2006.

[3] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70-78, Jan. 2002.

[4] P.P. Pande et al., "Design, Synthesis, and Test of Network On Chips," *IEEE Des. Test. Comput.*, vol 22. no. 5, pp. 404-413, Sept./Oct. 2005.

[5] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks—An Engineering Approach*, Morgan Kaufmann, 2002.

[6] S. Kumar et al., "A Network on Chip Architecture and Design Methodology," in *Proc. ISVLSI'02*, pp. 117-124, 2002.

[7] W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Packet-Switched Interconnections," in *proc. DAC'01*, 2001, pp. 683-689, 2001.

[8] F. Karim, A. Nguyen, and S. Dey, "An Interconnect Architecture for Networking Systems on Chips," *IEEE Micro*, Vol. 22, No. 5, pp. 36-45, Sept./Oct. 2002.

[9] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. DATE'00*, pp. 250-256, Mar. 2000.

[10] P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a Switch for Network on Chip Applications," in *Proc. ISCAS'03*, vol 5. pp. 217-220, May. 2003

[11] N. Honarmand, A.Shahabi, H. Sohofi, M. Abbaspour, and Z. Navabi, "High Level Synthesis of Degradable ASICs Using Virtual Binding," *VLSI Test Symposium*, 2007, in press.

[12] M. Yang, T. Li, Y. Jiang, and Y. Yang, "Fault-Tolerant Routing Schemes in RDT(2,2,1)," in *Proc. ISPAN'05*, pp. 1-6, Dec. 2005.

[13] R. Marculescu, "Networks-on-Chip: The Quest for On-Chip Fault-Tolerant Communication," in *Proc. ISVLSI'03*, pp. 8-12, Feb. 2003.

[14] C. Grecu et al., "On-line Fault Detection and Location for NoC interconnects," in *Proc. IOLTS'06*, pp. 145-150, July 2006.

Table1: Experimental Results

	# of Faulty Ports	Hard + Conceivable Soft Constraints				Hard Constraints			
		Longest Path Length	Average Path Length	Maximum Load	Average Load	Longest Path Length	Average Path Length	Maximum Load	Average Load
XY	0	6(1.00)	2.67(1.00)	16(1.00)	13.3(1.00)	6(1.00)	2.67(1.00)	16(1.00)	13.3(1.00)
EX1	1 ~ 2	6(1.00)	2.69(1.01)	25(1.56)	13.7(1.03)	6(1.00)	2.71(1.01)	28(1.75)	13.9(1.04)
EX2	3 ~ 6	7(1.17)	2.81(1.05)	34(2.13)	15.0(1.13)	7(1.17)	2.88(1.08)	36(2.25)	15.4(1.15)
EX3	3 ~ 6	6(1.00)	2.76(1.03)	32(2.00)	14.8(1.11)	6(1.00)	2.78(1.04)	32(2.00)	14.9(1.11)
EX4	5 ~ 10	7(1.17)	2.91(1.09)	46(2.88)	16.2(1.22)	8(1.33)	3.01(1.13)	50(2.13)	16.8(1.26)
EX5	8 ~ 16	9(1.50)	3.52(1.32)	40(2.50)	21.1(1.58)	9(1.50)	3.52(1.32)	40(2.50)	21.1(1.58)
EX6	9 ~ 18	7(1.17)	3.18(1.19)	46(2.88)	19.5(1.47)	8(1.33)	3.25(1.22)	50(3.13)	20.0(1.50)
EX7	11 ~ 22	8(1.33)	3.12(1.17)	41(2.56)	20.2(1.52)	8(1.33)	3.15(1.18)	44(2.75)	20.4(1.53)
EX8	14 ~ 28	9(1.50)	3.93(1.47)	64(4.00)	27.8(2.08)	9(1.50)	3.93(1.47)	64(4.00)	27.8(2.08)
EX9	15 ~ 30	9(1.50)	3.35(1.25)	70(4.38)	24.4(1.83)	9(1.50)	3.35(1.25)	70(4.38)	24.4(1.83)
EX10	18 ~ 36	9(1.50)	3.80(1.42)	64(4.00)	30.4(2.28)	9(1.50)	3.80(1.42)	64(4.00)	30.4(2.28)