Programmable Routing Tables for Degradable Torus-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 communication links in NoCs with torus topology. The method is based on the use of programmable routing tables in network switches. We investigate this technique for a conventional routing mechanism and our optimized routing mechanism. 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.**

I. INTRODUCTION

System-on-chip (SoC) design methodologies [1] have provided the appropriate and integrated solutions to manage the ever-increasing complexity of state-of-the art digital and mixed-signal systems. But still communication between processing elements remains an important challenge in SoC design methodology. Networks on Chips (NoCs) [2], [3] have been proposed as a solution to this problem. 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 [4] is commonly used in the NoC architecture as the switching scheme. Several NoC architectures and their implementation details are presented in [5], [6].

As the IC fabrication technologies move to nano-scale feature sizes and complexity and transistor count of the ICs grow, the produced dies become more and more susceptible to manufacturing faults and the process yield decreases [7]. Because 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 dies reusable. Such failures can occur in PEs and/or in communication fabric. Several methods have addressed this problem for the processing elements [8] and communication fabric [9], [10].

One possible method to work around the permanent faults is to use fault tolerant architectures. Marculescu [9] has discussed the possibility of achieving on-chip fault-tolerant communication based on a randomized gossip protocol in which 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 in software or hardware.

Yang et al. [10] have proposed a new torus [5]-like network and a fault tolerant routing algorithm to compensate the effect of link/node failures. 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 having the work done using the remaining non-faulty ones is another approach, usually referred to as Degradability [8]. In these techniques, first the faulty elements should be diagnosed. Then, using the built-in provisions in the circuit, the hardware could be reconfigured to use only non-faulty elements. The reconfigured circuit is likely to have a degraded performance, compared to the non-faulty one, because of the decreased number of available resources.

In this paper, we consider the problem of faulty links for a network with torus topology and present a degradable NoC structure to cope with such faults. In Section II, we propose a new routing mechanism for torus-based networks instead of the conventional routing mechanism which has serious drawbacks. Then, in Section III.A and III.B, we provide algorithms to reconfigure a network with faulty links that uses the conventional routing mechanism and our proposed routing mechanism, respectively. Finally, in Section IV, we compare the performance parameters of the reconfigured networks with faulty links against those of original non-faulty network.

II. SWITCHES AND ROUTING MECHANISMS

In NoCs, 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.

One important, and usually overlooked, aspect of the switch design is the implementation of the routing algorithm. 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 modified to cope with the link failures. Thus, the chip will become unusable whenever some links go faulty. But, in the case of PRTs, the switch can be reconfigured to bypass the broken links and get the job done using the remaining links. This approach, inevitably, adds an additional step of programming the PRTs to the manufacturing process. But, this will be of no considerable overhead

1-4244-0921-7/07 $25.00 © 2007 IEEE.

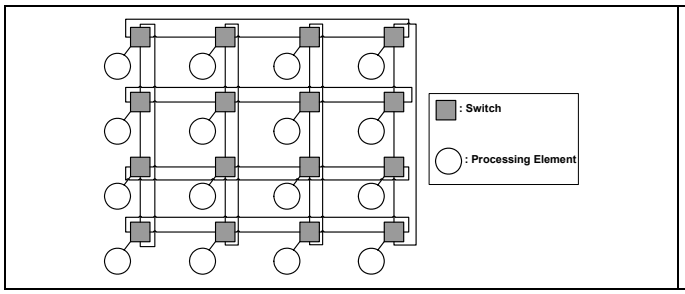


Figure 1. 2-D torus topology

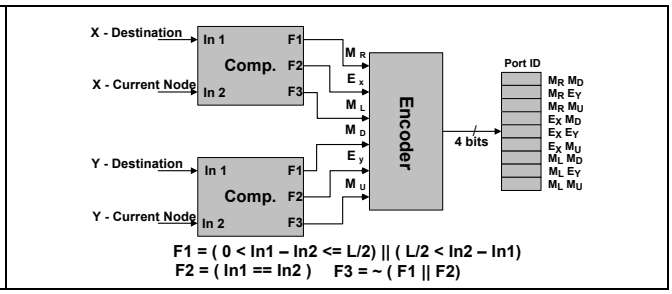


Figure 2. Torus-based routing mechanism

because mechanisms like JTAG or scan chains, which are widely employed for test purposes, can be used for this.

The paths taken by packets between source and destination switches, hence the contents of the PRTs, are defined by the routing algorithm in use. The algorithm must prevent live-lock [4] situations. A live-lock occurs when some packets will not reach their destination, even if they never get blocked permanently. Also, the routing algorithm should adhere to the interconnect topology and the addressing scheme in use.

The simplest way to implement a 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 contents of the PRT should be programmed according to the routing algorithm of the available topology.

But PAR is not the only possible, or even the best, implementation of PRTs. In fact, PAR suffers from two major drawbacks: (1) The size of the lookup table will grow linearly with the number of NoC nodes. Besides the area overhead, since table lookup should take place at least once for every received packet, the large size of the table and the resulting delay of lookup operation will decrease the switch throughput. (2) The switch cannot be used in networks with more nodes than the number of lookup table entries, and thus redesign will be needed. In other words, the PAR method is not amenable to network scaling.

Based on the particular topology and routing algorithm in use, there may be some methods to work around the above problems. In this paper we consider networks with 2-D torus topology (Fig. 1). In a 2D-Torus architecture [5], every switch is connected to four adjacent switches and to the related PE. The switches at the edges are connected to the switches at the opposite edge through wrap-around channels. In torus-based networks, the address of a node is a pair *(x,y)* which is the coordinates of the node in the torus. Each torus switch, in general, has five ports: one attached to the local PE (local port) and the other four to the neighboring switches (system ports). When a packet arrives, it should be delivered to the attached PE, through the local port, if it is destined for that node. Otherwise, one of the system ports (Left, Right, Up and Down) should be selected according to the destination address. Routing in the torus topology can be done with the modified version of XY algorithm [4], which was proposed for the mesh structure. In this algorithm, called Torus-XY, A packet is first routed in X direction if the *x*-coordinate of the destination is not equal with that of the current node. Choosing left or right port depends on the horizontal difference between the destination and the current node. 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, Fig. 2 shows our proposed routing mechanism. Two comparators compare the *x*- and *y*-coordinates of the current node with those of the destination. In this figure, the dimension of the torus is shown By *L*. The outputs of each comparator can assume three different one-hot coded states. For each comparator, *F1* output is activated in two cases: (1) when the difference between first and second input is between *0* (exclusive) and *L/2* (inclusive). (2) When subtracting *In1* from *In2* generates a value greater than *L/2*. *F2* output gets activated when the inputs are equal and finally when both of the *F1* and *F2* outputs are inactive, *F3* will be activated. Thus, we can have 9 different situations for the combination of comparator outputs. An encoder will generate a 4-bit signal indicating which of these 9 situations has occurred. The output of the encoder will be 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. Apparently, 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 Torus-Based Routing (TBR) because it is tailored to the torus topologies.

III. PROGRAMMING PRTS UNDER LINK FAILURES

The network topology could be considered as a graph with switches being its vertices and links being its edges. If this graph is connected, then the network will be **structurally connected**. We say that the network is **routing-connected** if for every source and destination pair of nodes *(SRC, DST)*, a packet originated 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 could be found to make the network routing-connected. We call a network **routing-connectable** if there is a set of PRT configurations to render the network routing-connected.

A. 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 such as path length and network congestion. Algorithms should be developed to obtain optimized routing table configurations if one is willing to use PAR-based routing tables in his/her design.

B. Programming TBR-based PRTs

It is possible for a network of TBR-based PRTs to be structurally connected but not routing-connectable. Fig. 3(a) shows an example of such a network. In this figure, the crosses indicate broken links. In this network, no TBR-based PRT configuration can be found for node B, because when B has a packet destined for F, it should send it using BF link and when it has a packet destined for J it should not use BF link. Thus, there is no feasible port identifier for (E_X,M_D) entry of B's routing table.

When a packet arrives at a switch, either it is destined for the attached processing element or it should 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

$$dist(cur, dst) = min(|x_\Delta|, |L - x_\Delta|) + min(|y_\Delta|, |L - y_\Delta|)$$

in which $x_\Delta = x_{dst} - x_{cur}$ and $y_\Delta = y_{dst} - y_{cur}$ and L is the dimension of the torus.

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 Torus-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 torus-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. A TBR-based routing table has one entry for every possible combination of positive moves, as shown in Fig. 2. 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, (M_U, M_L) in Fig. 2, 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 node becomes faulty, it needs some help from one of the neighboring nodes (helping node) to route the packet. Since the switch with a faulty link might be forced to perform a negative move, the distance may increase and if the helping node does not have its priorities properly set, it might return the packet to the original switch and cause live-lock. Hence, it would be necessary to put some constraints on the possible priority combinations that the helping node can use.

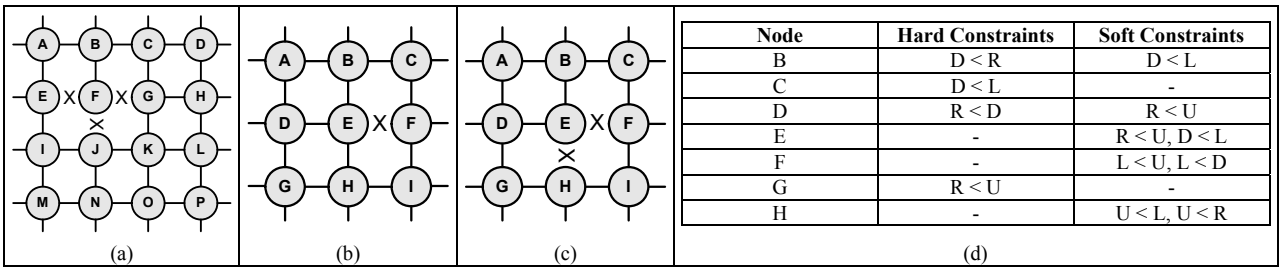
```
CONFIGURENETWORK(failure_set, current_constraints)
begin
  if ISEMPTY(failure_set) then
    for each configuration conf which satisfies all the costraints
      test all possible (src, dst) node pairs for a live-lock free path;
      if the test succeeds then
        return conf;
      end if;
    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 4. Reconfiguration algorithm for TBR-based routing tables

An example with one faulty link. To demonstrate our technique, Fig. 3(b) shows a torus structure with one broken link between nodes E and F. Since the EF link is faulty, Node E can select one of its three adjacent nodes (B, D, and H) as the helping node. Suppose that we choose node 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 EF link is faulty, 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 links and extract all the required constraints, it might be the case that for a switch, the constraints are conflicting, *e.g.*, a switch with both L<R and R<L constraints. Suppose that in our example, we first choose B as the helping node of E and after considering other nodes, 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 the 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 type of constraints, called Soft



Node	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 3. (a) a network which is not routing connectable using TBR-based routing tables (b) a network with one faulty link (c) a network with two faulty links (d) routing constraints in network of (c)

Constraint (SC) to reduce the average length of paths in the torus structure under faulty conditions. Suppose that in Fig 2(b), E is helped by node 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 node B to route when no other choice exists. Thus we consider the following SCs for node 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 can be removed. As another example, Fig. 3(c) shows a network with two faulty links and Fig 3(d) shows the assigned hard and soft constraints for each node. The nodes missing in the table have no constraints.

Reconfiguration Algorithm. Fig. 4 shows the reconfiguration algorithm. The CONFIGURENETWORK routine has two inputs: set of faulty links that have not been handled yet, and the constraints resulting from handling previous faults. It selects one faulty link from the list and, one by one, examines all possible helping nodes for the node affected by the faulty link. It repeats this process until there remains no faulty link 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 link. If there is no unchecked alternative, the search fails and the algorithm will return NULL to indicate failure.

IV. EXPERIMENTAL RESULTS

To assess the proposed technique, we have considered 10 examples 4 by 4 networks, with 1 to 11 faulty links. 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 I presents the results obtained for TBR-based technique. The row indicated with "Torus-XY" in the table gives the performance parameters for a network with no faulty links in which all the routing tables are configured according to Torus-XY routing mechanism. The values in parenthesis are normalized with regard to the corresponding values of non-faulty Torus-XY network.

As TABLE I shows, the general trend of performance parameters is to get worse when the number of faulty links 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 links in the network and the fact that the algorithm does search for first, not the best, possible solution. EX2 and EX3 demonstrate the sensitivity to pattern of faulty links. In both cases there are 2 faulty links and only the positions of the faulty links differ in two cases. As expected, when the algorithm is able to incorporate SCs, the results will be better than when only HCs are considered.

V. CONCLUSIONS

In this paper, we have considered the problem of faulty links in NoCs and have proposed a method for reconfiguring the routing mechanism of a network in order to bypass the faulty links. This method is based on using network switches with programmable routing tables. The reconfigured network might have a lower performance due to decreased number of communication links. The experimental results show that the algorithm could reconfigure the faulty networks to achieve acceptable performance parameters with regard to non-faulty networks using Torus-XY routing mechanism.

ACKNOWLEDGMENT

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

REFERENCES

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

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

[3] P.P. Pande C. Grecu, A. Ivanov. R. Saleh, and G. De Micheli, "Design, Synthesis, and Test of Network On Chips," *IEEE Des. Test. Comput*., vol 22. no. 5, pp. 404-413, Sept./Oct. 2005.

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

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

[6] P. P. Pande et al., "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025-1040, Aug. 2002.

[7] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, Edition 2005, 2005.

[8] 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.

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

[10] M. Yang, T. Li, Y. Jiang, and Y. Yang, "Fault-Tolerant Routing Schemes in RDT(2,2,1)/a-Based Interconnection Network for Networks-on-Chip Designs," in *Proc. ISPAN*'05, pp. 1-6, Dec. 2005.

TABLE I. EXPERIMENTAL RESULTS

	# of Faulty Links	Hard + Soft Constraints				Hard Constraints			
		Longest Path Length	Average Path Length	Maximum Load	Average Load	Longest Path Length	Average Path Length	Maximum Load	Average Load
Torus-XY	0	4	2.13	21	8	4	2.13	21	8
Exp1	1	4(1.00)	2.18(1.02)	24(1.14)	8.45(1.06)	5(1.25)	2.28(1.07)	36(1.71)	8.83(1.10)
Exp2	2	6(1.50)	2.32(1.09)	28(1.33)	9.27(1.16)	7(1.75)	2.40(1.13)	34(1.62)	9.60(1.20)
Exp3	2	6(1.50)	2.40(1.13)	24(1.14)	9.6(1.20)	6(1.50)	2.40(1.13)	24(1.14)	9.60(1.20)
Exp4	4	7(1.75)	2.60(1.22)	36(1.71)	11.1(1.39)	7(1.75)	2.60(1.22)	36(1.71)	11.1(1.39)
Exp5	5	8(2.00)	2.72(1.28)	35(1.67)	12.1(1.51)	10(2.50)	2.76(1.30)	35(1.67)	12.3(1.54)
Exp6	6	6(1.50)	2.78(1.31)	26(1.24)	12.9(1.61)	8(2.00)	2.88(1.35)	29(1.38)	13.3(1.66)
Exp7	7	7(1.75)	3.03(1.42)	48(2.29)	14.6(1.82)	7(1.75)	3.10(1.46)	48(2.29)	14.9(1.86)
Exp8	9	7(1.75)	2.98(1.40)	48(2.29)	15.8(1.95)	8(2.00)	3.13(1.47)	56(2.67)	16.4(2.04)
Exp9	10	7(1.75)	3.00(1.41)	50(2.38)	16.4(2.05)	8(2.00)	3.08(1.45)	56(2.67)	16.9(2.10)
Exp10	11	7(1.75)	3.15(1.48)	50(2.38)	18.0(2.25)	8(2.00)	3.23(1.52)	56(2.67)	18.5(2.31)