A New Scalable Fault Tolerant Routing Algorithm for Networks-on-Chip

Hamed Sajjadi Kia[1], Cristinel Ababei[2], Sudarshan Srinivasan[1], and Shaista Jabeen[1]
[1]Department of Electrical and Computer Engineering
North Dakota State University, Fargo ND, 58102 USA
Email: {hamed.sajjadikia, sudarshan.srinivasan, shaista.jabeen}@ndsu.edu
[2]Electrical and Computer Engineering
Marquette University, Milwaukee, WI, 53233, USA
Email: cristinel.ababei@marquette.edu

***Abstract*—In this paper we propose a distributed routing algorithm for networks-on-chip (NoCs) that can dynamically detect permanent failures in NoC links and recalculate routing paths using healthy links. What sets the proposed methodology apart from the previous works is that it provides a better tradeoff point between the improvement in fault tolerance and performance penalty due to the required redundancy and extra logic. An NoC prototype is implemented and simulated in Verilog-HDL to show the correct operation of the proposed adaptive routing.**

I. Introduction

Continuous shrinking of the feature size in deep-submicron domains has resulted in an increase in the adverse effects of process variations and aging mechanisms. These adverse effects translate into reliability issues in both components of multi-processor systems on chip (MPSoCs): processing cores and Network-on-Chip (NoC). Reliability has become a first class design concern aside from the traditional objectives that include performance, power dissipation, and area cost [1]. To deal with the reliability challenge, previous research has mostly focused on the processing elements of MPSoCs. However, the NoC as the communication unit, represents a significant portion of MPSoCs and only recently, reliability of NoC is being addressed. However, it is still largely an ongoing problem.

In this paper, we propose a new fault-tolerant adaptive routing algorithm for NoC that compared to previous works explores a larger solution space of possible routing scenarios. This improvement comes with an increase in the complexity of the routing algorithm. One of the main contributions of our new routing algorithm is the way we address this complexity. As an additional difference compared to previous works, the need to run the routing algorithm each time the system enters test mode is now eliminated. This helps to reduce the amount of time required for test mode and the power consumed during the process of updating the routing tables.

II. Related Work

Adaptive routing has been investigated as a primary mechanism to mitigate congestion and improve network performance. Many of these algorithms borrow ideas from routing in computer networks. For example, DyAD [2] is a combination of a static algorithm called oe-fix and an adaptive algorithm based on the turn-even model, while [3] is based on deflection routing. Other previous adaptive routing algorithms include region-based routing (RBR) [4], default-backup path (DBP) [5], and Vicis architecture [6].

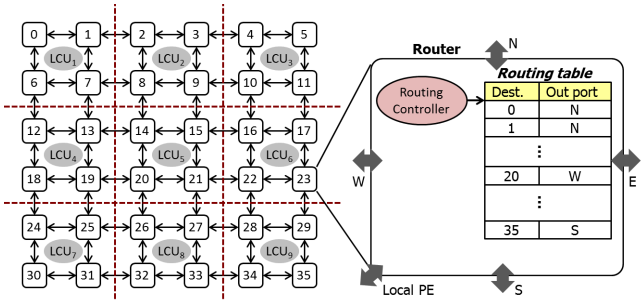

Fig. 1. Illustration of partitioning of a given NoC architecture into regions. Each region is managed by a local control unit (LCU).

Generally, routing algorithms can be classified as distributed or centralized. Centralized algorithms [7] utilize a central manager that collects information on the status of the links and routers. The manager computes region boundaries, new routing paths, various set-up control signals, etc., to reflect changes in network traffic. The advantage of this approach is that the manager always has a global view of the network. However, additional global control signals are required for collecting information about the network status and to update routing tables. Distributed algorithms [8], [9], on the other hand, do not use a central manager [10]. Routing decisions are made by local routing controllers typically based on congestion or stress information about the neighboring routers.

III. Proposed Distributed Adaptive Routing

Fault tolerance via adaptive routing comes at the expense of an increase in area and power consumption required by the additional hardware. To minimize this penalty, we design our adaptive algorithm as a distributed routing approach. We partition the NoC architecture into several regions. Each region has an associated local control unit (LCU), which is responsible with the routing activities of all packets that enter

any router contained within the region. The NoC example in Fig. 1 is partitioned into 9 equally sized regions. These regions do not need to be of equal area. We will present our discussion however in the situation when all partitions are equal for convenience and ease of explanations.

Each LCU is responsible with the routing of data to routers in its designated region and to the first-order neighboring routers adjacent to their designated region. For example in Fig.1, LCU_5 is responsible for routing packets that enter the region formed by the routers $\{14, 15, 20, 21\}$. The origin of the packets can be either the PEs connected to these routers or the routers in the neighboring adjacent regions.

Once a packet is injected into a router at one of its input ports, its corresponding output port is determined by simply reading from the routing table also located inside the router. When links suffer from permanent faults, routing paths are recalculated and routing tables are updated with new routing paths information. The updates are done by the routing controllers located inside each router. The routing paths recalculation is performed by LCUs. This is done by switching the whole system periodically to a testing mode state or on demand by the detection of a new hardware fault. Initially, when the network is hardware fault free, all routing tables implement a traditional static XY routing algorithm. As the system ages and hardware faults occur, the adaptive routing algorithm is triggered and new routing paths are calculated.

A. Description of the Local Control Unit (LCU)

Once the system enters the test mode, the status of each link is determined by an error detection mechanism integrated within each link and the results are reported to LCUs. For this purpose we use the error detection mechanism proposed in [11]. Each LCU uses the acquired data about the status of the 16 links in its designated region to determine the output port for the packets that enter the routers in its region. In our design, a given LCU does not need to know the status of the links that enter its designated region from neighboring regions; hence, the total number of links an LCU has to monitor is 16.

To more easily describe how an LCU and the adaptive routing algorithm work, let us focus our discussion on $router_{14}$, which is part of the partition controlled by LCU_5 of the NoC illustrated in Fig.1. Fig.2 describes the routing algorithm to determine the output port for the packets injected to $router_{14}$ from either PE_{14} or any of the neighboring routers based on their destinations and the status of the links in the region. Note that similar algorithms are run at the level of routers 15, 20, and 21. Also, note that although there are 16 links in each region, we only need the information about the status of 9 of them to determine the output port for the packets injected to each router in each region. These 9 links for $router_{14}$ are shown in Fig.3. The information about the status of the rest of the links in this region (to run the algorithm for $router_{14}$) is redundant. For example, the links that enter $router_{14}$ from the south and the east in this figure can bring data to it. However, since we are interested in routing data out of $router_{14}$ their status is not considered in the routing algorithm of $router_{14}$.

The algorithm from Fig.2 is rather complex from the point of view of required area and power overheads if implemented in hardware. Therefore, to address this issue, we run the algorithm offline and save the results in a look up table. This look up table will then be later utilized online directly by LCU_5 to update the routing tables located inside the routers that it monitors.

When a failure occurs in a link, a failure signal will be set to indicate the link failure. We refer to this signal as $error_i$. For example, an assertion of the $error_1$ signal means that link 1 is broken. These error signals indicate the status of all links in the network and are used by the routing algorithm in deciding the output ports in a given router where packets can be forwarded. They are also used as the address bits to indicate the location inside the look up table where the results of running the algorithm offline are stored. When the number and location of failures is such that LCUs cannot route packets toward their destinations, a system failure signal is asserted to signal that the system cannot be utilized any more.

Also each routing path should satisfy the following condition: packets should always be routed toward their destination in either X or Y direction. Packets are routed first in the X direction, unless that is not possible due to failed links or because the packets are at a router that already has the same Y coordinate as the destination. In such cases, packets are attempted to be routed in the Y direction. If that is not possible, the LCU sets the system failure signal.

Based on the (X,Y) location of destinations, all possible destinations are divided into 9 groups (see Fig.2). For example, group 1 includes all routers that are in the same row or column as $router_{14}$. Group 2 includes the routers whose X coordinate is smaller than the X coordinate of $router_{14}$ and their Y coordinate is larger than the Y coordinate of $router_{14}$ (routers $\{0, 1, 6, 7\}$ in Fig.1).

The algorithm from Fig.2 has primarily two for-loops. It starts with the first destination $j = 0$ ($router_0$) under the assumption that all 9 links (Fig.3) are functional $i = 0$, and repeats the process for all destinations ($j = 0...(N-1)$). Once the loop is completed, the results are saved in the first row of the look up table in the following order: first 3 bits of the first row of the look up table hold the output port id for packets that enter $router_{14}$, and whose destination is $router_0$ (under the assumption that all 9 links are functional). The 4th bit indicates if the system can identify a routing path for these packets or not. In this case, because all of the 9 links are functional it is set to 0. The routing information for the remaining destinations is saved in the same order in the first row of the look up table − as shown in Fig.2. The above process is repeated under the assumption that only link 0 is broken ($i = 1$ or $i[0] = 1$). The corresponding routing information is saved in the second row of the look up table. This process is repeated until we generate and save the routing information for all possible destinations under all link failure situations.

The operation of the routing algorithm is further described in the following scenario examples. Consider a situation where a packet arrives to $router_{14}$ and its destination is $router_{17}$.

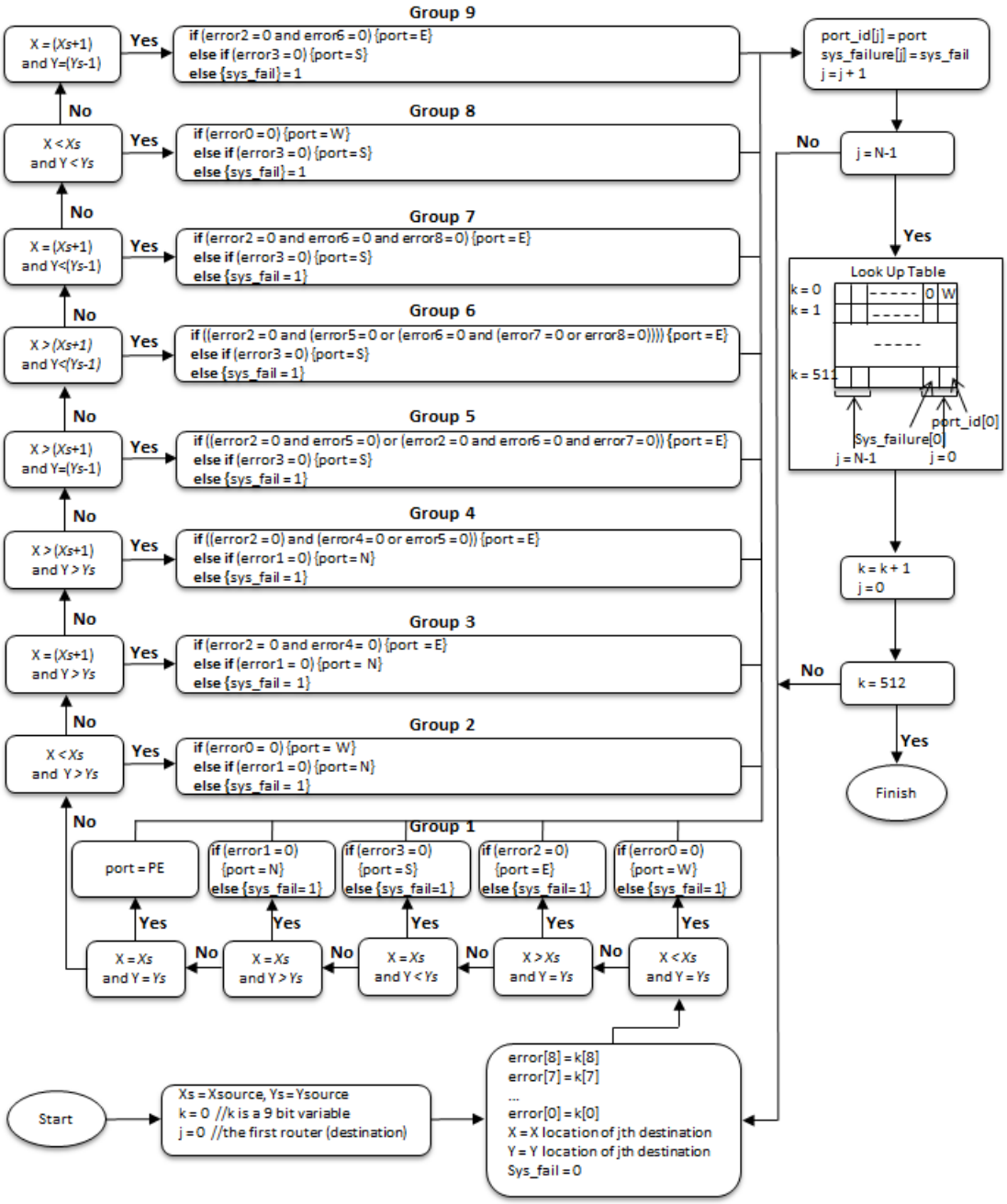

Fig. 2. Block diagram of the proposed adaptive routing algorithm for $router_{14}$.

$router_{17}$ is in the first destination group. Based on the assumption that packets should always move toward their destination either in X or Y direction, we see that there is only one path to the destination. The algorithm checks the status of link 2 (Fig.3). If it is healthy, the packet will be routed to the east output port (to $router_{15}$). At this stage, the algorithm will not check the status of link 5. Once the packet enters $router_{15}$, the algorithm for $router_{15}$ will check the status of link 5 and if it is broken it will generate a system failure signal. As another example, let us consider the source destination pair, $router_{14}$ and $router_5$ in Fig.1. $router_5$ is located in group 4 ($X_{router_5} > (X_{router_{14}}+1)$ and $Y_{router_5} > Y_{router_{14}}$ (Fig.2)). Because the destination is in region 3, the LCU_5 should send the packets to one of the routers $\{8, 9, 16\}$ in regions 2 and 6. The routing algorithm first tries the X direction. If link 2 and one of the links 4 or 5 (Fig. 3) are healthy, the packets will be routed to the east output port (this is because once the packets enter $router_{15}$ there will be at least one path open to send the packets toward region 3). Otherwise, the algorithm will check the status of link 1. If it is healthy, packets will be

routed to the north output port. Otherwise the system failure signal will be asserted.

As mentioned earlier, the routing algorithm is run offline for all destinations under all possible link failure scenarios and the results are recorded in a look up table (as is illustrated in Fig. 2). Therefore, there is no need to run the algorithm again and update the tables. As already described, the error bits are utilized to address different locations in the look up table. This technique effectively allows the LCUs to get pre-computed routing information easily under different link failure situations. Fig.4 illustrates the process of updating the routing tables by the routing controllers integrated inside the routers using the look up tables in LCUs.

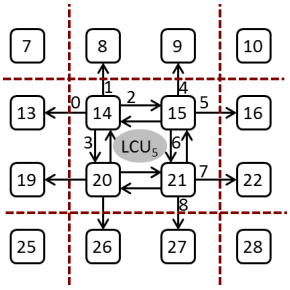

Fig. 3. The 9 links used for routing data for $router_{14}$.

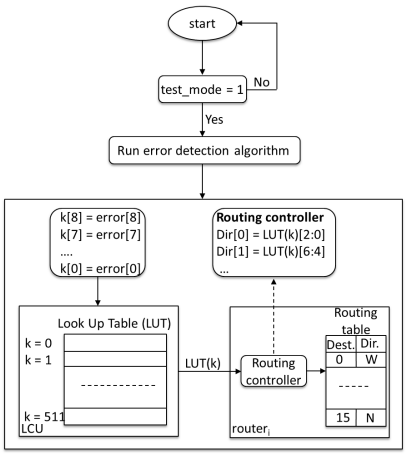

Fig. 4. Illustration of the process of updating the routing tables.

IV. Experimental results

We developed and implemented a complete regular mesh NoC that has no support for adaptive routing and compared our proposed NoC against it. Both these NoC architectures are coded in Verilog, synthesized and simulated with Xilinx ISE WebPack. We find that the proposed custom NoC requires 5.1% more area compared to the traditional NoC, it consumes on average 7.1% more power. The maximum achievable operation frequency is only 3.86% lower, compared to the traditional XY routing algorithm. Table I depicts a comparison of the area and power overheads between the proposed fault tolerant algorithm and some of the previously proposed fault tolerant routing algorithms.

TABLE I
Area and power overhead of different fault tolerant routing algorithms for NoC.

Fault tolerant routing algorithm	Area overhead (compared to XY algorithm)	Power overhead (compared to XY algorithm)
Proposed adaptive routing algorithm	5.1%	7.1%
DyRS-NM [12]	5.64%	NA
DSPIN [8]	8%	NA
FT_XY3 [13]	6.1%	5.2%
Negative first [14]	33.2%	NA

V. Conclusion

We proposed a fault tolerance oriented adaptive routing algorithm for regular networks-on-chip. The proposed algorithm helps to provide improved fault tolerance and graceful performance degradation in the face of increasingly adverse hardware faults due to ageing mechanisms. A 4×4 NoC prototype implemented on a Virtex-5 FPGA validated the correct operation of the proposed routing algorithm. In addition, the hardware implementation allowed us to realistically estimate the performance penalty due to the extra hardware.

References

[1] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, and L.S. Peh, Research challenges for on-chip interconnection networks, IEEE Micro, 27 (5) (2007) 96-108.

[2] J. Hu and R. Marculescu, DyAD - smart routing for Networks-on-Chip, Proc. Design Automation Conf., 2004, pp. 260-263.

[3] A. Kohler and M. Radetzki, Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches, Proc. 3rd Int. Symp. on Networks-on-Chip, 2009, pp. 22-31.

[4] J. Flich, A. Mejia, P. Lopez, and J. Duato, Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chip, Proc. first Int. Symp. on Networks-on-Chip, 2007, pp. 183-194.

[5] M. Koibuchi, H. Matsutani, H. Amano, and T. Pinkston, A Lightweight Fault-tolerant Mechanism for Network-on-chip, Proc. second Int. Symp. on Networks-on-Chip, 2008, pp. 13-22.

[6] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, A highly resilient routing algorithm for fault-tolerant NoCs, Proc. Design Automation and Test in Europe, 2009, pp. 21-26.

[7] T. Skeie, F.O.S. Jacobsen, S. Rodrigo, J. Flich, D.Bertozzi, and S. Medardoni, Flexible DOR Routing for Virtualization of Multicore Chips, Proc. Int. Symp. on System-on-Chip, 2009, pp. 073-076.

[8] Z. Zhang, A. Greiner, and S. Taktak, A reconfigurable routing algorithm fora fault-tolerant 2D-mesh Network-on-Chip, Proc. Design Automation Conf., 2008, pp. 441-446.

[9] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos, A highly robust distributed fault-tolerant routing algorithm for NoCs with localized rerouting, Proc. Workshop on Interconnection Network Architecture, 2012, pp. 29-32.

[10] M. Li, Q.A. Zeng, and W.B. Jone, DyXY - a proximity congestion-aware deadlock free dynamic routing method for Network-on-Chip, Proc. Design Automation Conf., 2006, pp. 849-852.

[11] H.S. Kia and C. Ababei, Improving fault tolerance of Network-on-Chip links via minimal redundancy and reconfiguration, Proc. Int. Conf. on Reconfigurable Computing and FPGAs, 2011, pp. 363-368.

[12] F. Ge, N. Wu, and Y. Wan, A Network Monitor based Dynamic Routing Scheme for Network on Chip, Proc. Asia Pacific Conf. on Postgraduate Research in Microelectronics and Electronics, 2009, pp. 133-136.

[13] M. Valinataj, S. Mohammadi, and S. Safari, Fault-aware and Reconfigurable Routing Algorithms for Networks-on-Chip, IETE Journal of Research, 57 (3) (2011) 215-224.

[14] H. Zhu, P.P. Pande, and C. Grecu, Performance evaluation of adaptive routing algorithms for achieving fault tolerance in NoC fabrics, Proc. Int. Conf. on Application Specific Systems, Architectures and Processors, 2007, pp. 42-47.