Fault-Tolerant Topology Generation Method for Application-Specific Network-on-Chips

Suleyman Tosun, Vahid B. Ajabshir, Ozge Mercanoglu, and Ozcan Ozturk

***Abstract*—As the technology sizes of integrated circuits (ICs) scale down rapidly, current transistor densities on chips dramatically increase. While nanometer feature sizes allow denser chip designs in each technology generation, fabricated ICs become more susceptible to wear-outs, causing operation failure. Even a single link failure within an on-chip fabric can halt communication between application blocks, which makes the entire chip useless. In this paper, we aim to make faulty chips designed with network-on-chip (NoC) communication usable. Specifically, we present fault-tolerant irregular topology-generation method for application-specific NoC designs. Designed NoC topology allows different routing path if there is a link failure on the default routing path. Additionally, we present a simulated annealing-based application mapping algorithm aiming to minimize total energy consumption of the NoC design. We compare fault-tolerant topologies with nonfault-tolerant application-specific irregular topologies on energy consumption, performance, and area using multimedia benchmarks and custom-generated graphs. Our results demonstrate that our method is able to determine fault-tolerant topologies with negligible area increase and better energy values.**

***Index Terms*—Energy minimization, fault tolerance, mapping, network-on-chip (NoC), topology design.**

I. Introduction

TECHNOLOGY improvements have made it possible to place millions of transistors on a single chip, resulting in more complex and denser designs than ever. Now, designers can embed all system components on one chip, which is called system-on-chip (SoC). Traditional SoC design space exploration has focused on computational aspects of the applications whereas today's design efforts have shifted toward communication-based design space exploration as a result of increase in the number of SoC components. Thus, the communication architecture plays a major role in performance, area, and energy consumption of the SoC designs. Previous studies show that traditional shared-bus and point-to-point-based architectures do not scale well for current communication intense applications [1]. As a solution to this scalability problem, researchers introduced a new on-chip communication architecture, called network-on-chip (NoC) [2], [3].

NoC architectures can be constructed using regular or irregular topologies. Although regular topologies are easy to construct and reusable, applications cannot be well optimized on them. Irregular topologies are beneficial for designing application-specific NoCs since the design parameters such as energy consumption, performance, and area can be optimized better than their regular counterparts [4]. Several studies have been published regarding energy-efficient and/or fault-tolerant regular topology-based NoC designs, especially for mesh topologies [5]–[7]. However, studies of irregular application-specific topologies are restricted to explore energy efficiency [8]; fault tolerance has not been considered.

Application-specific topologies generated by current methods have only one communication path between any communicating nodes [4], [8], [9]. If there is a permanent fault in any of the links or ports as a result of the fabrication process, the system cannot recover its functionality and the chip becomes useless. Motivated by this fact, in this paper, we propose a fault-tolerant application-specific topology generation method for NoC-based designs. In our method, we first generate random nonfault-tolerant irregular isomorphic topologies. We then add extra links to the generated topologies to make them fault-tolerant. Our method generates a topology library (TL) consisting several fault-tolerant topology alternatives and a ring topology, which is also a fault-tolerant topology with minimum number of network resources. The designer can select a topology from the generated TL that meets the design goals.

Our fault-tolerant NoC designs can be used for various fault scenarios.

1) Designs with at least one link failure due to the CMOS fabrication process can still be used, albeit with a gracefully degraded performance.
2) Designs with at least one link failure during the lifetime of the chip due to electromigration can still operate at a gracefully degraded performance level.
3) Dynamically adaptive routing mechanisms can easily be added to the generated fault-tolerant topologies, which enables the application to use several routing alternatives.

In this paper, we also propose an application mapping algorithm that can be applied to fault-tolerant and nonfault-tolerant NoC designs. The objective of the proposed mapping

Manuscript received June 30, 2014; revised October 7, 2014, December 25, 2014, and February 12, 2015; accepted February 23, 2015. Date of publication March 16, 2015; date of current version August 18, 2015. This work was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 112E360, and in part by EU COST Action IC1204—TRUDEVICE. This paper was recommended by Associate Editor S. Kim.

S. Tosun is with the Department of Computer Engineering, Hacettepe University, Ankara 06800, Turkey (e-mail: stosun@hacettepe.edu.tr).

V. B. Ajabshir and O. Mercanoglu are with the Department of Computer Engineering, Ankara University, Ankara 06500, Turkey.

O. Ozturk is with the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey.

Digital Object Identifier 10.1109/TCAD.2015.2413848

algorithm is to minimize energy consumption while meeting bandwidth constraints. Our mapping algorithm is based on the commonly used simulated annealing (SA)-based method [10]. Since the mapping problem is NP-complete, SA is a good choice for this kind of problem. SA is a probabilistic method for finding the global optimum of a cost function. It is modeled on the behavior of condensed matter at low temperatures. The annealing process starts with an initial configuration and continuously reduces the temperature of the system in search of a better configuration. If the process obtains a better energy values in a configuration, it directly accepts this configuration. Otherwise, it accepts the new solution based on an acceptance probability function. In this way, it makes uphill moves from a local minimum in an attempt to jump to a valley where a global minimum might reside.

We compare our fault-tolerant topologies with nonfault-tolerant counterparts based on energy consumption, area, and performance. Our results show that with a small area increase, our method generates fault-tolerant topologies with better energy values.

This paper is organized as follows. In Section II, we briefly review the related work on fault-tolerant and nonfault-tolerant topology designs for NoCs. We explain the motivation of this paper in Section III, and we introduce the problems we deal with in Section IV. In Sections V and VI, we present our fault-tolerant topology generation (FTTG) algorithm and our SA-based mapping algorithm, respectively. In Section VII, we show the experimental results. Finally, in Section VIII, we conclude this paper.

II. Related Work

In this section, we examine the related work in two parts. In the first part, we list the previous work on irregular topology generation methods that do not consider fault tolerance. In the second part, we present the related work that consider reliability and fault tolerance for irregular topologies, which is the main focus of this paper.

A. Irregular Topologies

One of the pioneering work aiming to generate irregular topologies for NoC architecture designs is presented in [9], and applies a two-step approach for designing the topology. In the first step, it uses integer linear programming (ILP)-based method for floor planning the application nodes and the NoC components on the given chip area. In the second part, it generates the topology by determining the connections between the NoC parts and the application nodes. Even though this paper shows promising results, it also shows that ILP-based methods take too much CPU time to find an optimal solution. The same research group thus developed a faster method for the same problem using genetic algorithms [11]. In both methods, the authors aimed to reduce the energy consumption of the final application-specific irregular-topology-based NoC architecture.

There are also heuristic-based irregular topology generation algorithms in [4] and [8]. These methods first determine the topology connection and then later decide the floorplanning, as opposed to the methods in [9] and [11]. To obtain the topology, they also use a two-step approach. In the first step, they cluster the application nodes by placing heavily communicating nodes together. In the second step, they determine how to connect these clusters on the routers of the topology.

Although all these methods obtain very promising results when considering energy minimization of the generated irregular topology, they do not consider fault tolerance in the design. If any link has a permanent fault, the produced chip becomes useless.

B. Fault-Tolerant NoC Design

There have been some efforts at NoC designs that do consider fault tolerance. For example, Dally *et al*. [12] introduced the reliable interconnection design concept. In this paper, the authors propose a reliable router architecture that transmits data over the network even if there is a transient fault in the links. Similar work focuses on the router architecture for reliable network transmission [13], introducing an architecture called BulletProof, which uses N-modular redundancy. The basic idea of this paper is to implement multiple (generally three) copies of the same router and check the packet sent by each router. It then decides on the correct version of the result based on majority voting. A similar redundancy-based technique uses duplication [14], doubling the network components and determining fault in the system by checking both copies of the sent packets. Although above mentioned redundancy-based methods can be used for generating fault-tolerant irregular topologies, they increase the number of network components tremendously, resulting in high chip area and energy consumption overheads.

The methods mentioned above consider transient fault tolerance. Early work on permanent fault tolerance generally study regular (especially mesh-based) topologies. If a router or a link on a mesh-based NoC has a permanent fault, such studies focus on routing methods that avoid sending packets over the faulty part [15]–[22].

To increase NoC designs' robustness, some prior efforts focus on reconfigurability of NoCs [23], [24]. For example, the MADNESS project [23] aims to design NoC architecture with adaptive fault-tolerant capabilities. Cannella *et al.* [24] presented a method that moves the job of the faulty core to the neighboring core during run time.

This paper focuses on tolerating permanent link and router port failures. Prior work also aims to tolerate link failures [25]–[31], proposing rerouting mechanisms after locating faulty links.

All the aforementioned studies focus only mesh-based or regular topology-based NoCs and there are more routing path options for these topology types. However, the current irregular topologies uses only one path between any communicating nodes; thus, the methods for regular topologies cannot be applied for tolerating permanent link failures on irregular topologies.

In this paper, we propose a new idea to generate fault-tolerant application-specific irregular topology design. We aim to add a minimum number of extra routers and links to the

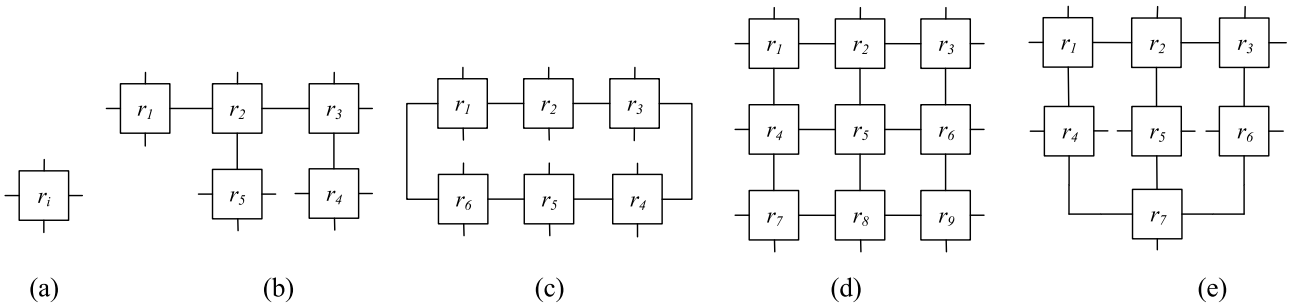


Fig. 1. Different topology examples for an application with 12 nodes. (a) Router with four ports. (b) Irregular topology with no fault-tolerance capability. (c) Ring, (d) mesh, and (e) fault-tolerant irregular topologies.

topology to achieve fault-tolerance capability. If the produced chip is not faulty, it works with the default routing. If there is an error in any link, we can use an alternative routing with only a small degradation in application performance. The new routing requires at least two alternative paths between any router pair. In this paper, our goal is to design such a topology with the least area increase and at least two paths between routers exist. We present the preliminary version of this paper in [32]. This paper extends our preliminary version by several aspects. We examine the previous work in detail and motivate the importance of the studied problem. We present the energy model and mapping problem. We give in depth analysis of the proposed FTTG algorithm and present our SA-based mapping method. Contrary to previous version, we conduct our experiments with newer technology parameters and give more experimental evaluations. Finally, we give a routing scenario in this version of this paper.

III. Motivation

When we design an NoC architecture for a given application, we must first select the system topology. For this step, we have the option to select either a regular or irregular topology. The selection criteria can be based on energy consumption, performance, throughput, fault tolerance capability, and/or chip area.

In Fig. 1(a), we show an abstract view of a four-port router used in our topologies. Each link connected to a router port is assumed to be bi-directional (i.e., each port can be used as input or output). In our topology design, we use homogeneous routers (i.e., all routers have the same number of ports) to better deal with design complexity and to maintain regularity. Fig. 1(b)–(e) shows, respectively, examples of an irregular topology with no fault-tolerance capability, a ring topology, a mesh topology, and a fault-tolerant irregular topology. These topologies can be used for an application with up to 12 nodes because at most 12 empty ports are available for each topology. Ring and mesh are two examples of regular topologies.

As seen in Fig. 1, each topology has a different number of routers. Nonfault-tolerant irregular topology has the smallest number of routers because it is configured to minimize NoC energy consumption and has no fault-tolerance capability. The remaining three topologies have fault-tolerance capability because an alternative communication path between any router pair exists even if there is a permanent link failure.

Mesh and ring topologies are two basic regular topology examples that can tolerate at least one link failure. However, a link failure's effects on each topology type can be different. While a link failure on a ring topology may result in very high energy consumption and performance degradation in new routing because there is only one alternative rerouting path, a mesh topology has more than one rerouting option and thus can offer better energy and performance values.

In this paper, we investigate an irregular topology option for fault-tolerant NoC design. To achieve this goal, we generate an application-specific topology with two alternative paths between any routers in the topology. If a permanent link failure is detected on the design, we select an alternative routing for the application with only a small degradation in both energy consumption and performance. Our goal is to achieve a better energy consumption and area overhead than the mesh topology counterpart. To do so, we add extra routers and links to nonfault-tolerant irregular topologies.

Even though we design the topology to tolerate single link failure the designed topology can tolerate multiple link errors only if removing the faulty links from the topology does not disconnect network components. If removing the faulty links from our topology disconnects the network, the designed topology cannot be used. One way to overcome this problem and cover multiple link failures can be doubling the links. However, when we double all the links, each router will need extra ports, which increase the network area and energy consumption. Thus, we aim to tolerate the single link failure that may occur with higher probability than multiple link failures.

IV. Problem Definition

In this section, we first present the energy model used in this paper and then define the fault-tolerant application-specific topology generation and mapping problems for NoC architecture.

A. Energy Model

The energy consumption of the functional blocks (i.e., the cores) and the network resources (i.e., the routers and links) constitute the total energy consumption of NoC architecture. In this paper, our goal is not to minimize the energy consumed by the cores, but to try minimizing the energy consumed by the network components. The network's energy consumption is

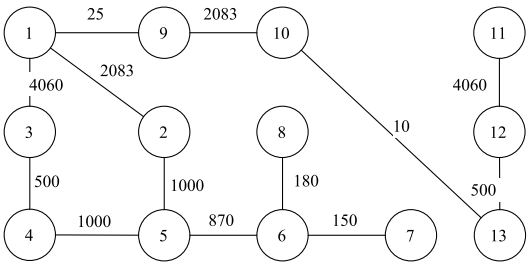


Fig. 2. CFG of MP3 encoder.

TABLE I
SHORTEST PATH VALUES OF THE GRAPH GIVEN IN FIG. 3(a)

r	1	2	3	4	5	6	7	8
1	0							
2	1	0						
3	2	1	0					
4	2	1	1	0				
5	1	2	**3**	**3**	0			
6	2	1	2	2	1	0		
7	**3**	2	**3**	2	2	1	0	
8	**3**	2	2	1	**3**	2	1	0

directly proportional to the amount of bit transmissions on the network. To estimate the energy consumption of NoC architecture, we should use an energy model based on the total bit transmissions. We give the total energy consumption of one bit ($E_{T_{\text{bit}}}$)

$$E_{T_{\text{bit}}} = E_{R_{\text{bit}}} + E_{L_{\text{bit}}} \tag{1}$$

where $E_{R_{\text{bit}}}$ and $E_{L_{\text{bit}}}$ represent the energy consumption of the routers and the links, respectively. The average energy consumption of sending one bit data from core v_i to v_j can be calculated as

$$E_{T_{\text{bit}}}^{v_i,v_j} = \eta_{v_i,v_j} \times E_{R_{\text{bit}}} + \delta_{v_i,v_j} \times E_{L_{\text{bit}}} \tag{2}$$

where η_{v_i,v_j} is the number of routers the bit passes through and δ_{v_i,v_j} is the link length in millimeter between the source and destination routers. Since the link lengths between routers are not known before floor planning, we assume fixed link lengths for the topology generation step based on the chip dimensions.

B. Problem Definition

Our goals for the topology generation problem are: 1) to determine a topology such that all communicating cores of the application can transmit data to each other over the network with at least two alternative paths and 2) to minimize energy consumption. To achieve these goals, the number of routers for the system must first be determined. Then, the resultant topology must ensure that each router can be reached from all other network routers via two paths and that all the cores are connected to at most one router port. Additionally, the routing must be deadlock and network-congestion free (i.e., the router port and link bandwidth requirements must be satisfied.) To explain this problem more formally, we give the following definitions.

Definition 1: A core flow graph (CFG) is a graph $G(V, E)$ where each vertex $v_i \in V$ represents a core (i.e., a node) in the application, and each edge $e_{i,j} \in E$ represents a dependency between two tasks v_i and v_j. The amount of data transfer between v_i and v_j is represented by the weight $w_{i,j}$ for all $e_{i,j}$ and is given in bits per second.

In Fig. 2, we give the CFG of the MP3 encoder, taken from [9].

Definition 2: A topology graph (TG) is a connected graph $T(R, L)$ where R represents the set of routers and L represents the set of links connecting the routers.

In Fig. 3(a) and (b), we give two examples of a TG. In both topologies, two alternative paths between any router pair exist. In the first topology, we use 8 routers and 9 links, whereas in the second we use 9 routers and 11 links.

Graph diameter and average path length (APL) of the network are the two factors affecting the system's total communication. Thus, we try to minimize these two parameters in the generated topology.

Definition 3: Graph diameter d_T of a TG $T(R, L)$ is the maximum of the shortest distances between all pairs of the vertices (i.e., routers), and can be calculated as

$$d_T = \max\{d(r_i, r_j)\}, \forall(r_i, r_j) \in R \tag{3}$$

where $d(r_i, r_j)$ represents the shortest distance between vertices r_i and r_j.

To calculate the diameter of the TG, we first find all shortest paths between each pair of vertices. Then, we select the maximum distance. In Table I, we give the shortest paths of the graph in Fig. 3(a). The graph diameter of this topology is 3.

Definition 4: APL_T of a topology T is the average of the shortest paths between any pairs of the vertices of the TG. Let r denote the number of vertices of the given topology. Then, the APL_T is calculated by the following formula:

$$\text{APL}_T = \frac{2}{r(r-1)} \sum_{r_i \le r_j} d(r_i, r_j). \tag{4}$$

For example, the APL of the graph in Fig. 3(a) is 1.92, while the APL of the graph in Fig. 3(b) is 1.86.

Problem 1 (FTTG): Given a set of nodes n and the set of routers, each having p ports, determine the number of routers (r) and the number of links (l) to generate the topology. Then generate the topology that meets the following criteria.

1) *Constraints:*
   a) The topology must be fully connected with the set of routing paths P, where each path $p_{i,j}$ is the routing path between each pair of routers (r_i, r_j).
   b) For each path $p_{i,j}$, each link $l_{k,l}$ on this path should satisfy the bandwidth constraint $\text{bw}(l_{l,k})$.
   c) Additionally, to satisfy the fault-tolerance criteria, there must be at least two alternative routing paths between any router pairs. That is

$$\forall(r_i, r_j) \in R, (p_{i,j}) \ge 2. \tag{5}$$

2) *Objective Function:*
   a) The objective function of the FTTG problem is to minimize the APL of the generated topology T.

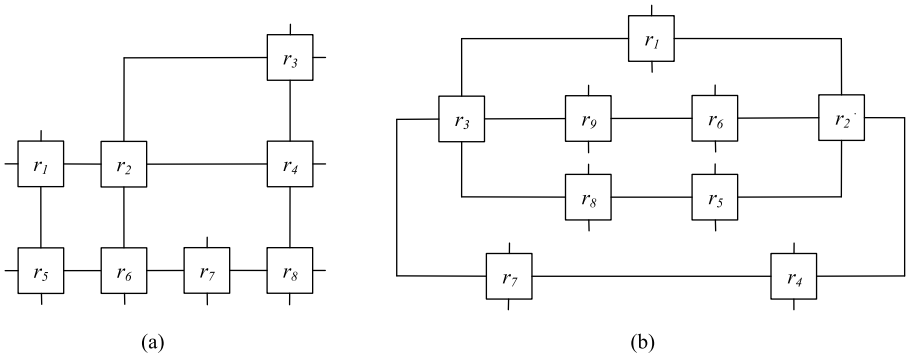


Fig. 3. Two examples of a fault-tolerant TG. (a) TG with 8 routers and 9 links and (b) 9 routers and 11 links.

In other words, our objective function is

$$\min : \mathrm{APL}_T. \tag{6}$$

Problem 2 (Application Mapping): Given a CFG $G(V, E)$ and a TG $T(R, L)$, determine a mapping such that each node of the CFG is mapped to a router on the TG and the total energy consumption of the designed NoC is minimized. We mathematically formulate the application mapping problem as follows.

Given a CFG and a TG that satisfy

$$|V| \leq |R(p_e)| \tag{7}$$

where $|V|$ is the number of vertices of the given CFG and $|R(p_e)|$ is the number of empty ports of the topology routers, find a many-to-one mapping function $M : V \rightarrow R$ from the CFG to the TG with

$$\min : E_{\mathrm{NoC}} = \sum_{\forall e_{i,j} \in E} w_{i,j} \times E_{T_{\mathrm{bit}}}^{v_i, v_j} \tag{8}$$

such that

$$\forall v_i \in V, \exists r_k(p_e) \in T, \quad M(v_i) = r_k(p_e) \tag{9}$$

$$\forall v_i \neq v_j \in V, \quad M(v_i) \neq M(v_j) \tag{10}$$

where E_{NoC} is the total energy consumption of the network and $r_k(p_e)$ represents the empty port p_e of router r_k. Since some routers may have more than one empty port, the mapping function has a many-to-one relation.

V. FTTG Algorithm

Before explaining the details of our FTTG algorithm, we give an overview of our approach in the following section.

A. Overview of FTTG Algorithm

We give the flowchart of the FTTG algorithm in Fig. 4. Our algorithm has two main phases: 1) generating nonfault-tolerant irregular topology using a minimum number of routers and links and 2) adding extra routers and links to obtain a fault-tolerant version of the topology.

As shown in the flowchart, our method accepts the number of nodes (n) of the given application (CFG), the number of ports (p) for routers, and the iteration count (t) as inputs. Based on these input values, it generates a ring topology, which is a fault-tolerant topology with minimum numbers of routers and links. It then adds the ring topology to the TL as our first topology. After that, it calculates the minimum number of routers ($r_{\min}$) and links ($l_{\min}$) for non-FTTG (N-FTTG) and the maximum number of routers ($r_{\max}$) and links ($l_{\max}$) that will be used for fault-tolerant topologies. Since the fault-tolerant irregular topology must utilize at least $r_{\min}$ routers, we start FTTG with $r_{\min}$ routers. At each outer loop of the FTTG algorithm, we add one more router to the routers at hand until we reach $r_{\max}$.

After selecting the number of routers (r) for the topology, we determine how many extra links can be added to the network using the formula $l = l_{\max} - l_{\min}$. When the routers connected with links to generate a topology, enough empty ports must be left for the application nodes, which is n here. We then generate a random, fully connected, nonfault-tolerant topology with r routers and $r - 1$ links. Certainly, some of the routers must be connected to other routers with at most one port. Thus, we connect these routers to each other by adding l links, aiming to minimize the APL of the topology. Each router and link must be on a cycle to have at least two alternative routing paths, thus our next step is to check that this is so. If there is a fault-tolerant topology with r ports in the TL, we check whether the newly generated topology is isomorphic with the existing topologies. If it is, we simply discard the new topology; otherwise, we add it to the library. This topology generation process iterates t times, which is a predefined iteration count.

In our output TL, we may have several topology alternatives with different numbers of routers, varying between $r_{\min}$ and $r_{\max}$. The designer can select any of these topologies that suits the design objectives, or the one with the minimum APL. In the following sections, we give the details of each step in the FTTG algorithm.

B. Calculating the Required Number of Network Components

For irregular N-FTTG and for regular and irregular FTTG, we first need to determine the minimum number of routers and links. We give the necessary calculations for nonfault-tolerant irregular topology, ring topology, and FTTG in the following paragraphs.

1) Nonfault-Tolerant Irregular Topology: Given an application with n nodes and a router set having p ports each,

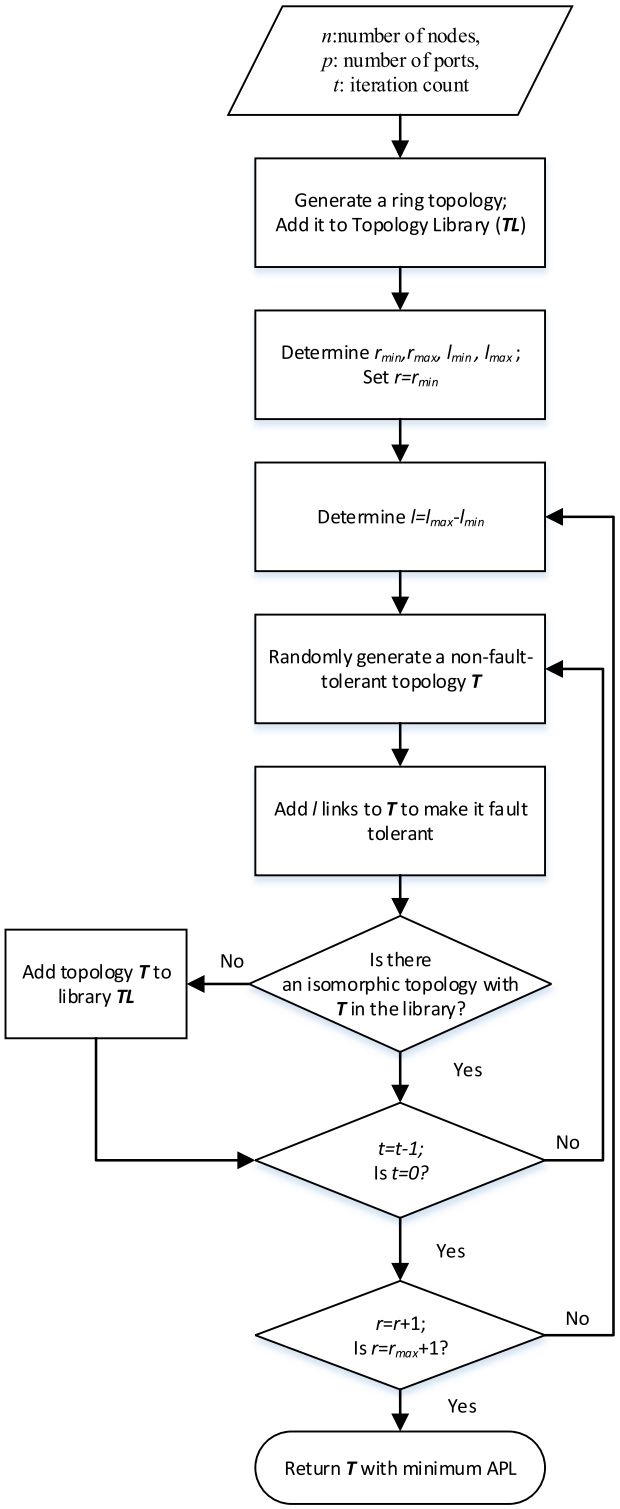


Fig. 4. Flowchart of FTTG algorithm.

where $p > 2$, we can determine the minimum number of routers $r_{\min}$ for nonfault-tolerant irregular topology generation using the following formula:

$$r_{\min} = \left\lceil \frac{n-2}{p-2} \right\rceil. \tag{11}$$

For $r_{\min}$ routers, each of which has p ports, we can have at most $pr_{\min}$ ports. Each link consumes two ports to connect two routers. For $r_{\min}$ routers, we can have at least $r_{\min}-1$ links, which means all the links consume at least $2(r_{\min}-1)$ ports. Then, the remaining ports can be used for connecting at most $pr_{\min} - 2(r_{\min}-1)$ nodes. This inequality can be written as

$$n \le pr_{\min} - 2(r_{\min}-1). \tag{12}$$

Solving this inequality for $r_{\min}$ gives us (11). Clearly, the generated nonfault-tolerant irregular topology will need at least $l_{\min}$ links to connect $r_{\min}$ routers, where $l_{\min} = r_{\min} - 1$, as proven in [4].

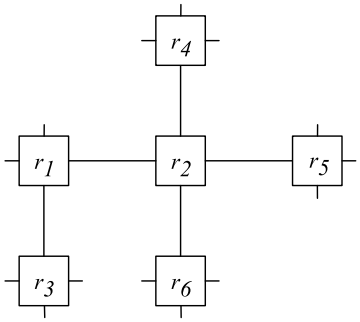


Fig. 5. Example of nonfault-tolerant irregular topology for an application with 14 nodes. The topology uses a minimum number of four port routers and links.

As an example, we calculate the minimum number of routers and links for an application with 14 nodes, assuming each router has four ports. When we use (11), we find $r_{\min} = 6$ and $l_{\min} = 5$. In Fig. 5, we show an example of a randomly generated nonfault-tolerant irregular topology using six routers with four ports and five links. In this topology, there exist 14 empty ports for 14 application nodes.

2) Ring Topology: Given a set of routers $r_{\min}$ with p ports, where $p > 2$, and a set of nodes n of the given application, the minimum number of routers $r_{\min}$ to generate a valid ring topology can be found by the following formula:

$$r_{\min} = \left\lceil \frac{n}{p-2} \right\rceil. \tag{13}$$

For $r_{\min}$ routers, each of which has p ports, we can have at most $pr_{\min}$ ports. Each link consumes two ports to connect two routers. For $r_{\min}$ routers, a ring topology needs $r_{\min}$ links. Thus, all links consume $2r_{\min}$ ports. Then, the remaining ports can be used, connecting $pr_{\min} - 2r_{\min}$ nodes. This inequality can be written as follows:

$$n \le pr_{\min} - 2r_{\min}. \tag{14}$$

Solving this inequality gives us (13).

3) Fault-Tolerant Irregular Topology: As stated above, the ring topology is a fault-tolerant regular topology because it has exactly two routing paths from any router pair. However, it has a high APL value when the number of routers in the topology is high. To reduce the APL of a fault-tolerant topology, we need to add extra links to minimize the graph diameter, resulting in a minimized APL. When we add extra links to the nonfault-tolerant irregular topology or to the ring topology, the required empty ports for application nodes will be reduced. Thus, we cannot map the given application's nodes

Algorithm 1: N-FTTG

Data: n: Number of nodes, p: Number of ports for routers.
Result: $T'(R', L')$: Generated topology, $\text{APL}_{T'}$, R^1: Routers with one link connection.

```
1  begin
2      Determine r_min using Equation (11);
3      Add r_min routers to R = {r_1, r_2, ..., r_min};
4      R' = ∅; L' = ∅;
5      Select a router r_1 ∈ R;
6      R = R − r_1; R' = r_1;
7      P' = p(r_1); /* Add empty ports p(r_1) to empty
           port list P'                                  */
8      for i = 2 to i = r_min do
9          Select router r_i ∈ R;
10         R = R − r_i; R' = R' + r_i;
11         Randomly select an empty port p(r_j) ∈ P';
12         Randomly select a port p(r_i) from r_i;
13         Connect r_i and r_j with the link l_{p(r_i),p(r_j)};
14         L' = L' + l_{p(r_i),p(r_j)};
15         P' = P' − p(r_j) + (P(r_i) − p(r_i)); /* Remove
               connected ports from and add empty
               ports of r_i to P'.                       */
16     Determine R^1 : routers with p − 1 empty ports;
17     Calculate APL_{T'};
18     return T'(R', L'), R^1, and APL_{T'};
```

on the topology. Therefore, we must add extra routers to the nonfault topology to be able to add more links.

One option to resolve this issue is to double the routers in the design. However, doing this will also double the area of the network. Therefore, we limit the number of additional routers to $\log_2(r_{\min})$ to have scalable increase for FTTG. We determine the maximum number of routers using the following formula:

$$r_{\max} = \lceil r_{\min} + \log_2(r_{\min}) \rceil. \tag{15}$$

When we utilize $r_{\max}$ routers, each having p ports, we need at least n empty ports to generate a topology. The remaining ports can be used for links to connect routers. Since we can have $pr_{\max}$ ports and each link consumes two ports, we can determine the maximum number of links $l_{\max}$ as follows:

$$n \leq pr_{\max} - 2l_{\max} \tag{16}$$

$$l_{\max} = \left\lfloor \frac{pr_{\max} - n}{2} \right\rfloor. \tag{17}$$

To generate a fault-tolerant irregular topology, we first start with $r_{\min}$ routers. We then increase the number of routers by one until we reach $r_{\max}$. Thus, the number of routers for the generated fault-tolerant topologies varies between $r_{\min}$ and $r_{\max}$. In the following paragraphs, we explain how we generate the topologies.

C. N-FTTG Algorithm

As we show in Fig. 4, after determining the required number of routers for both N-FTTG and FTTG, we randomly generate a nonfault-tolerant irregular topology using our N-FTTG algorithm. We give the sketch of N-FTTG in Algorithm 1.

In this algorithm, using (11), we first determine how many routers will be needed to generate a valid irregular topology. We then select the first router $r_1 \in R$ and move it to R'. After moving r_1 to R', we keep track of empty ports in the list P'. In the for loop of the N-FTTG algorithm, we select a router r_i from R and connect it to the router in R'. To do this, we randomly select a port from P' and a port from r_i and connect them. We accordingly update the router list R', link list L', and the empty port list P'. When we connect a router in R', we use one port from P' and one port from r_i. When we remove these ports from the empty port list, we add the empty ports of r_i to P'. We continue these random connections until $R = \emptyset$, which is $r_{\min} - 1$ times.

After the for loop terminates, we have a nonfault-tolerant, fully connected irregular topology, which has at least n empty ports to be mapped on. In this algorithm, we calculate the APL of the generated topology and the routers that are connected to the network with only one link, R^1, (i.e., the routers that have $p - 1$ empty ports). We use the list R^1 in our FTTG algorithm.

D. Adding Extra Links

The N-FTTG algorithm generates irregular topologies, in that there is only one routing path from one router to another. If there is a permanent fault on any of the links or ports on the specified path, the application may not operate properly. Therefore, the fabricated chip cannot be used. For example, if there is a failure in the topology given in Fig. 5, there will not be alternative path to send packets. For an alternative path between any communicating router pair, we must add extra links to the topology generated by the N-FTTG algorithm.

As shown in Fig. 4, after generating nonfault-tolerant topologies, we add extra links to the generated topology by leaving enough empty ports to map n nodes of the given application. We give the sketch of this process in Algorithm 2, where we first determine how many extra links can be added to the topology. As stated above, nonfault-tolerant topology uses $l_{\min} = r - 1$ links. Then, using (17), we then determine the maximum number of links $l_{\max}$ that can be used in the fault-tolerant topology. Clearly, we can add $l = l_{\max} - l_{\min}$ links to the topology at hand. To do this, we add l links one by one by connecting selected router pairs.

When we select the pairs to be connected, we use the router set R^1, which contains routers that have only one link connection to the network, so that we can have one more alternative port connection to the remainder of the network. Additionally, all the network components (i.e., routers and links) must be on a cycle in the generated topology to be fault-tolerant. If there are at least two routers in R^1, we select the router pairs from this set. However, we can have more than one option for this selection. In this case, we select two routers r_i and r_j from R^1 with a maximum distance d_{r_i,r_j} to minimize the APL_T of the topology T. If we have one router in R^1, we select this router and another router in the topology with the maximum distance. If $R^1 = \emptyset$ and we still have additional links that can be added to the topology, we select two routers from the topology to minimize APL_T. After each selection, we connect the selected routers r_i and r_j with a new link l_{r_i,r_j}.

For example, the topology in Fig. 5 has four routers, $R^1 = \{r_3, r_4, r_5, r_6\}$, with only one link connection. Suppose we can

Algorithm 2: AddLinks

Data: n: Number of nodes, p: Number of ports for routers, $T'(R', L')$: Non-fault-tolerant topology, R^1: Routers with one link connection, TL: Topology library.
Result: TL: Updated topology library.

```
1  begin
2      Determine l_max using Equation (17);
3      l = l_max − l_min;
4      for i = 1 to l do
5          if |R^1| ≥ 2 then
6              Select r_i, r_j ∈ R^1 s.t. d_{r_i,r_j} is maximum;
7              R^1 = R^1 − r_i − r_j;
8          else if |R^1| = 1 then
9              Select r_i ∈ R^1 and r_j ∈ R' s.t. d_{r_i,r_j} is maximum
               and r_j has at least one empty port;
10             R^1 = R^1 − r_i;
11         else
12             Select r_i, r_j ∈ R' s.t. d_{r_i,r_j} is maximum and
               r_i and r_j have at least one empty port;
13         Connect r_i and r_j with l_{r_i,r_j};
14         L' = L' + l_{r_i,r_j};
15     isomorph = 0; cycle = 0;
16     if R^1 = ∅ ∧ min − cut(T') ≥ 2 then
17         cycle = 1;
18     for All topologies in TL do
19         Select a topology T(R, L) ∈ TL;
20         if T'(R', L') = T(R, L) then
21             isomorph = 1;
22     if isomorph = 0 ∧ cycle = 1 then
23         TL = TL + T'(R', L');
24     return TL;
```

add two extra links to this topology. In this topology, the maximum distance is $d_{\max} = d_{r_3,r_4} = d_{r_3,r_5} = d_{r_3,r_6} = 3$. We then randomly select r_3 and r_4 to connect. Then, R^1 becomes $R^1 = \{r_5, r_6\}$. For the second link, we select r_5 and r_6.

After connecting all extra links to the topology, we check that all the network components (i.e., routers and links) are on a cycle. We know that if $R^1 = \emptyset$ after the link connections, all routers are on a cycle. However, this may not be true for the links. Fig. 6 shows an example topology of all routers on a cycle but link l_{r_3,r_4} not on a cycle. If there is an error in this link, the system may not operate properly. If all the links in the topology are on a cycle, the minimum cut degree of the topology T, $\min - \text{cut}(T)$, must be at least 2. If $R^1 = \emptyset$ and $\min - \text{cut}(T) \geq 2$, we determine that all the components of the topology T are on a cycle. If there is even a single component not on a cycle, we simply discard the topology. We should note here that FTTG algorithm generates a valid topology if $R^1 = \emptyset$ and all network components are on a cycle; otherwise, it only returns the ring topology as our fault-tolerant topology.

After generating the fault-tolerant topology, we compare it with previously generated fault-tolerant topologies. We check whether the topology at hand is isomorphic with any topology in the TL. If it is, we discard the topology at hand; if it is not, we add it to the TL. We use a polynomial-time algorithm to check the isomorphism between two graphs [33]. In Fig. 7, we give two isomorphic graphs. In these graphs, router numbers are in different places but network connections are the same.

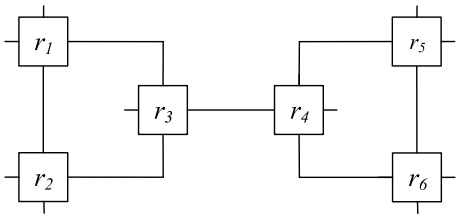


Fig. 6. Topology with a link not on a cycle.

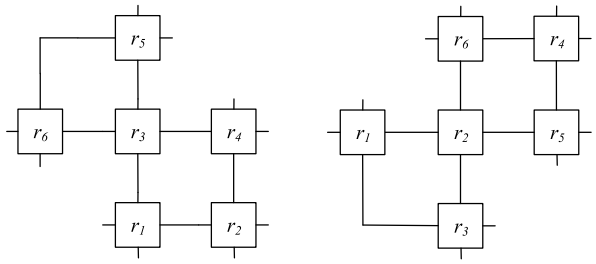


Fig. 7. Isomorphic graphs.

E. Complexity of FTTG

Given r routers, $l_{\min}$ links for N-FTTG, $l_{\max}$ links for FTTG, an application with n nodes, and an iteration count t, we calculate the complexity of the FTTG algorithm as follows.

If we exclude the time complexity of the APL calculation for the generated topology, the complexity of N-FTTG can be written as $O(r)$.

After generating the nonfault-tolerant topology $T(R, L)$, we add $l = l_{\max} - l_{\min}$ links to it. When we add each link, we calculate the path length for each router pair; its time complexity is $r^2(|L| + |R| \lg |R|)$, where $(|L| + |R| \log_2 |R|)$ is the complexity of Dijkstra's shortest path algorithm. Then, the time complexity of this process can be written as $O(lr^3)$ because $|R| = r$ and $r > |L|$.

We check the $\min - \text{cut}(T)$ in the polynomial time complexity of $O(r^3)$ using the Edmonds–Karp algorithm [34]. We then check if the generated topology is isomorphic with the ones in the TL. Assuming we have at most $t-1$ topologies with r nodes in the library, we use the algorithm presented in [33], which has a time complexity of $O(r^5)$. The outer loop of the FTTG algorithm runs $\log_2 r$ times. As a result, the time complexity of FTTG can be approximated as $O(tr^5 \log_2 r)$, which is dominated by the graph isomorphism part.

VI. Application Mapping Algorithm

After generating a set of fault-tolerant topologies and creating a TL, we select a topology with a minimum APL value for NoC design. The second phase of NoC design is mapping the application with the objective of energy minimization, as stated in Section IV.

We give the pseudo code of an SA-based application mapping algorithm in Algorithm 3, where we randomly map tasks onto the topology (line 2) to obtain an initial mapping. We then calculate the total energy consumption cost of the initial mapping using (8) and the shortest path routing.

Algorithm 3: SA-Based Mapping

```
Data: G(V, E), T(R, L), bw_min
Result: M: Mapping, E_NoC
1  begin
2      M = Random_Initial_Mapping(G, T)
3      C = Calculate_E_NoC(T)
4      M_best = M
5      C_best = C
6      Temperature = ⌈10 ln |R|⌉
7      for i → 0 to |R|^2 do
8          Reject = 0
9          while Reject < 10 do
10             M' = neighbor(M)
11             C' = Calculate_E_NoC(T')
12             bw_req = Calculate_max_bandwith_requirement(T')
               ΔC = C − C'
13             Generate a random variable α, 0 ≤ α ≤ 1
14             if bw_req ≤ bw_min ∧ (ΔC ≤ 0 or
               α ≤ e^(−ΔC)/Temperature) then
15                 M = M'
16                 Reject = 0
17             else
18                 Reject + +
19             if Reject = 0 ∧ C < C_best then
20                 M_best = M
21                 C_best = C
22         Decrement Temperature
23     M = M_best
24     Comm = C_best
25     return M, E_NoC
```

After the initial mapping is determined, we set temperature to its highest value. The temperature parameter in our mapping problem is analogous to the distance between two nodes mapped on the topology. If the distance between two exchange nodes is high, the temperature is high and vice versa. Marcon *et al.* [36] stated that the algorithm obtains good results when the initial temperature is selected to be $\lceil 10 \ln |R| \rceil$, where $|R|$ is the number of routers in the topology.

After the temperature of the system is initialized, the algorithm executes two nested loops. While the external loop searches for global minima, the internal loop tries to refine the local solution. We limit the number of external loop iterations to $|R|^2$, as suggested in [36]. The internal loop randomly selects two nodes and swaps them to determine a new solution. It then evaluates whether the new solution is better than the solution at hand. It also checks whether the bandwidth requirement $\mathrm{bw_{req}}$ of the generated topology is less than or equal to the allowed bandwidth limit $\mathrm{bw_{min}}$. If it is, the solution is accepted as the current solution. Otherwise, it generates a random variable α, where $0 \leq \alpha \leq 1$, and compares it with the acceptance probability function $e^{(-\Delta C)/\mathrm{temperature}}$. If the result of the function is higher than α, the new move is accepted. At high temperatures, the acceptance probability is also high. When we lower the temperature of the system, acceptance probability decreases. We limit the iteration of the internal loop to ten consecutive rejects. After each iteration, we decrement the temperature of the system and start a new iteration, accepting the solution at hand as our initial solution. Our SA-based algorithm returns the mapping with minimum energy consumption value.

VII. Experimental Results

In this section, we evaluate the FTTG algorithm by comparing the topologies generated by FTTG with the ones generated by N-FTTG. We implemented both algorithms in C++ and evaluated them using our implementations. In the first set of experiments, we compare the FTTG algorithm with N-FTTG based on APL and area under varying numbers of nodes. In the second set of experiments, we use real multimedia benchmarks and custom graphs to compare area, energy, and performance [i.e., average hop count (AHC)]. Finally, we give a case study that shows several alternative topologies and mappings for a benchmark example.

A. Evaluating FTTG

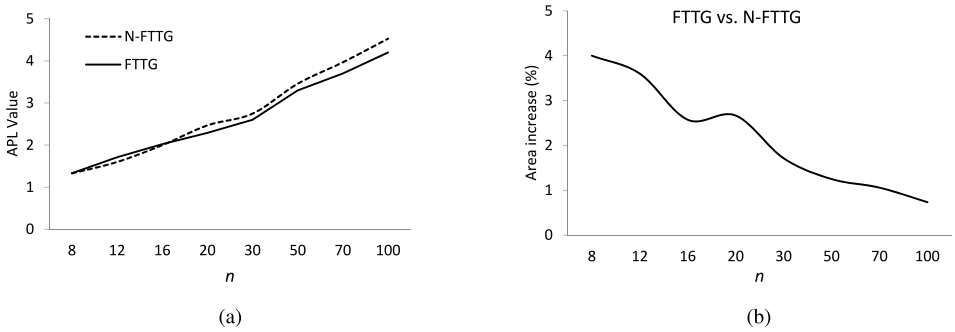
In this set of experiments, we generate topologies using FTTG and N-FTTG for applications with different numbers of nodes (n). We select the iteration count $t = 500$ and the number of router ports as 4, 5, and 8. We conduct experiments with node numbers between 8 and 100. We give the APL and area comparisons in Table II. In the table, we give the results for n varying only between 8 and 20 due to space concerns.

In Table II, we show the APL and area overhead brought by the FTTG algorithm against topologies generated by the N-FTTG algorithm. The first column gives the number of ports for the routers used in each topology generation. In the second column, we give the number of nodes that can be mapped on the generated topologies. The third and fourth columns show the APL and minimum number of routers used for nonfault-tolerant topologies. Note that we can generate more than one topology with N-FTTG algorithm, and thus we select the one with best APL value. Columns 5–7 show the APL, the number of routers, and links used for FTTG, respectively. As N-FTTG, FTTG generates several topology alternatives and we select the one with best APL value. While column eight shows the APL increase brought by the FTTG algorithm, the last column shows the area overhead of FTTG against N-FTTG. The negative values in column eight mean that FTTG generated topologies have better APL values. When we calculate the area overhead, we assume that the area of the network components for N-FTTG consume 6% of the total chip area, as suggested by Dally and Towles [2]. For example, when $n = 8$ and $p = 4$ in Table II, N-FTTG uses three routers. That means each router consumes 2% chip area. On the other hand, FTTG uses four routers, increasing the chip area by 2%, which is shown in the last column of Table II. In this calculation, we omit the area increase of the extra links as in [35] because the area consumed by the links is negligible when compared to the area consumed by the routers. The network area increase can be calculated separately using the number of routers for each topology.

As column 8 in Table II shows, FTTG generates topologies with close or better APL values than N-FTTG counterpart. When the number of nodes or the number ports of the router increases, the APL gain of FTTG against N-FTTG also increases. The importance of APL values is evident in the mapping phase. When the APL value is smaller, the AHC

TABLE II
APL AND AREA COMPARISON OF FTTG AGAINST N-FTTG WITH VARYING NUMBERS OF NODES AND PORTS

p	n	N-FTTG		FTTG			APL increase (%) of FTTG vs. N-FTTG	Area increase (%) of FTTG vs. N-FTTG
		APL	r	APL	r	l		
4	8	1.33	3	1.33	4	4	0	2
	12	1.60	5	1.71	7	8	6.87	2.4
	16	2.00	7	2.02	10	12	1.11	2.57
	20	2.47	9	2.29	13	16	-7.29	2.67
5	8	1.00	2	1.00	3	3	0	3
	12	1.50	4	1.33	4	4	-11.33	0
	16	1.60	5	1.57	8	12	-1.87	3.6
	20	1.67	6	1.71	8	10	2.39	2
8	8	0.00	1	0.00	1	0	0	0
	12	1.00	2	1.00	2	2	0	0
	16	1.33	3	1.00	3	3	-24.81	0
	20	1.33	3	1.00	4	6	-24.81	2



(a) (b)

Fig. 8. Comparison of FTTG with N-FTTG. (a) APL value comparison with varying numbers of nodes. (b) Area-increase percentages of FTTG against N-FTTG.

(and latency values) of the application will be smaller, which results in a better performance.

As the last column in Table II shows, the area increase brought by the FTTG algorithm is tolerable. For example, using four port routers, the area increase compared to the N-FTTG topologies is 2.41% on average. One important observation is that when the number of application nodes increases, the area overhead decreases. This result shows that the area increase for applications with large numbers of nodes and for topologies with large numbers of routers will be very small.

As stated above, the range of nodes in our experiments changes from 8 to 100. In Fig. 8, we illustrate how the APL values and area-increase percentages scale with varying numbers of nodes for topologies that use four port routers. In Fig. 8(a), we give the APL comparison for N-FTTG and FTTG. For this comparison, we select the topologies with the best APL values for N-FTTG and FTTG. As the APL values for the two types of topologies show, for a small number of nodes, our FTTG algorithm determines topologies with similar APL values to the N-FTTG. After the number of nodes exceeds 16, the FTTG determines better APL values than its counterpart.

We give the maximum area overhead in percentages in Fig. 8(b), comparing the area increase against N-FTTG. As the graph shows, the area overhead is within tolerable limits. In the area comparison, the fluctuation in part of the function is because the selected number of routers increases for varying n values. We select the maximum number of routers using (15), and because the $\log_2(r_{\min})$ increases when the number of nodes increases, fluctuations occur. As the graph in Fig. 8(b) shows, when the number of nodes increases, the area overhead decreases. For large numbers of nodes, the APL value increase of FTTG compared to N-FTTG is within tolerable limits. However, FTTG brings a fault-tolerance capability with a small area increase. With diminishing technology size, we can expect that the number of nodes for future applications will be much higher than now; therefore, our FTTG method will be much more effective in future while it still meets the current fault-tolerant topology needs.

B. Evaluating the Mapping Algorithm

In this set of experiments, we evaluate the FTTG and SA-based mapping algorithms on a set of multimedia benchmarks and custom graphs. We select six video applications from the literature as benchmarks, namely the video object plane decoder and the MPEG-4 decoder from [37], the multi-window display from [8], and the 263 decoder, 263 encoder, and MP3 encoder from [9]. Since the number of nodes for the selected benchmarks range between 12 and 16, we randomly generate three application graphs with higher numbers of nodes. We name these three graphs as G30, G40, and G50 and they have 30, 40, and 50 nodes, respectively.

For this set of experiments, we use four port routers. After determining the number of routers and links, we generate two

TABLE III
ENERGY, AREA, AND LATENCY COMPARISONS FOR N-FTTG AND FTTG METHODS AND MAPPINGS

Graph	n	Energy (mJ)		Energy increase (%) of FTTG against N-FTTG	Area Overhead (%) of FTTG against N-FTTG	Average Hop Count	
		N-FTTG	FTTG			N-FTTG	FTTG
MPEG-4	12	62.86	61.10	-2.80	1.2	1.23	1.23
VOPD	16	62.34	65.15	4.51	0.85	1.05	0.95
MWD	12	20.93	19.14	-8.55	1.2	0.67	0.58
263 Dec	14	0.266	0.297	11.65	1	0.73	0.93
263 Enc	12	3.94	3.74	-5.08	2.4	0.83	0.75
MP3 Enc	13	0.241	0.254	5.39	1	1.07	0.76
G30	30	31.03	29.15	-6.06	0.42	2.18	1.81
G40	40	52.05	50.86	-2.27	0.34	2.95	2.69
G50	50	44.05	43.62	-0.98	0.28	3.33	3.23

topology alternatives. For the N-FTTG and FTTG topologies, we select the one with a minimum APL value. We then map the applications onto the generated topologies using our SA method. In the mapping process, we aim to minimize only the dynamic energy consumption of the network components (i.e., the total energy consumption of sending data over routers and links). For our energy calculations, we use the energy model given in Section IV-A. For energy consumption parameters, we adopt the energy consumption values for routers and links for 45-nm technology given in [35] and we derive the energy values for 22-nm technology using the scaling factors from ITRS [38]. In 22-nm technology, we estimate the energy consumption of the routers at 3.20 pJ/Kb and the link energy consumption at 4.78 pJ/Kb/mm. We assume the length of the links as 1 mm.

We present the results of these experiments in Table III. In the first two columns, we give the name of the graph and the number of nodes for the given graph, respectively. Columns 3 and 4 give the energy consumptions of the mappings for the N-FTTG and FTTG, respectively. Column 5 shows the energy comparison of FTTG against N-FTTG. The negative values in this column means that mappings on FTTG topologies has better energy values than N-FTTG. Column 6 gives the area overhead of the FTTG topologies against N-FTTG topologies. Finally, the last two columns show the AHC value to compare the latency for the two mappings.

As the energy values for six benchmarks in Table III show, our FTTG and the mappings obtain very close energy values to the N-FTTG most of the time. In most of the cases, it obtains better results than its counterpart. The energy gain of FTTG against N-FTTG is 0.47% on average for this nine application graphs. We should note here that when the number of application nodes is small (i.e., less than 16) our FTTG algorithm returns a ring topology as the topology with best APL value. When the application nodes is high, it generates a fault-tolerant irregular topology that is different than ring. We show an example FTTG topology examples for MP3 encoder benchmark in the next section.

The area increase of FTTG topologies against N-FTTG is around 0.97% on average, which is in tolerable limits. The AHC values of FTTG topologies are better than N-FTTG most of the time. This means that the latency of the application running on FTTG topologies will be better than N-FFTG topologies.

The last three rows of Table III give the results of three mappings for custom generated graphs G30, G40, and G50, respectively. For all custom graphs, energy and AHC values of FTTG are better than N-FTTG and the area increases are less than 0.5%. This shows that FTTG performs better than N-FTTG when the number of application nodes increases.

As the set of experiments on real benchmarks demonstrates, our FTTG algorithms brings fault tolerance to NoC design, with only a small area overhead. When the number of application nodes increases, FTTG performs better than N-FTTG and it determines topologies with lower energy consumption than its counterpart. As we stated above, with diminishing technology size, we can expect that the number of nodes for future applications will be much higher than now; therefore, our FTTG method will be much more effective in future while it still meets the current fault-tolerant topology needs.

C. Case Study: MP3 Encoder

In this section, we illustrate the generated topologies and the mapping results for the benchmark MP3 encoder using four port routers. We give the generated topologies and the mapping results in Fig. 9. Encircled numbers in this figure represent the nodes of the MP3 encoder application. Note that the mappings may not be optimal because each mapping is obtained using our SA-based mapping algorithm, and SA-based methods do not guarantee determining optimal solution. In this figure, we illustrate the mapping on ring, N-FTTG, and two FTTG topologies, respectively. We compare these mappings based on area, energy consumption, and on AHC as the latency parameter.

The area increases of the first FTTG (eight routers) against the ring (seven routers) and N-FTTG (six routers) are 0.85% and 2%, respectively, while they are 1.71% and 3% for the second FTTG (nine routers). The AHC values for the four topologies in Fig. 9 are 0.76, 1.07, 0.84, and 1.08, respectively. The energy consumption of the ring and N-FTTG are calculated as 0.254 and 0.241 mJ, while they are 0.257 and 0.269 mJ for the FTTGs. As the energy values illustrate, the FTTG in Fig. 9(c) brings only 6.7442% energy overhead against the N-FTTG. Our FTTG given in Fig. 9(d) brings a 11.27% energy increase against the N-FTTG. Therefore, the designer's preferred topology would be the one in Fig. 9(a) because it achieves better energy values than the others and results in less area increase and better AHC values.

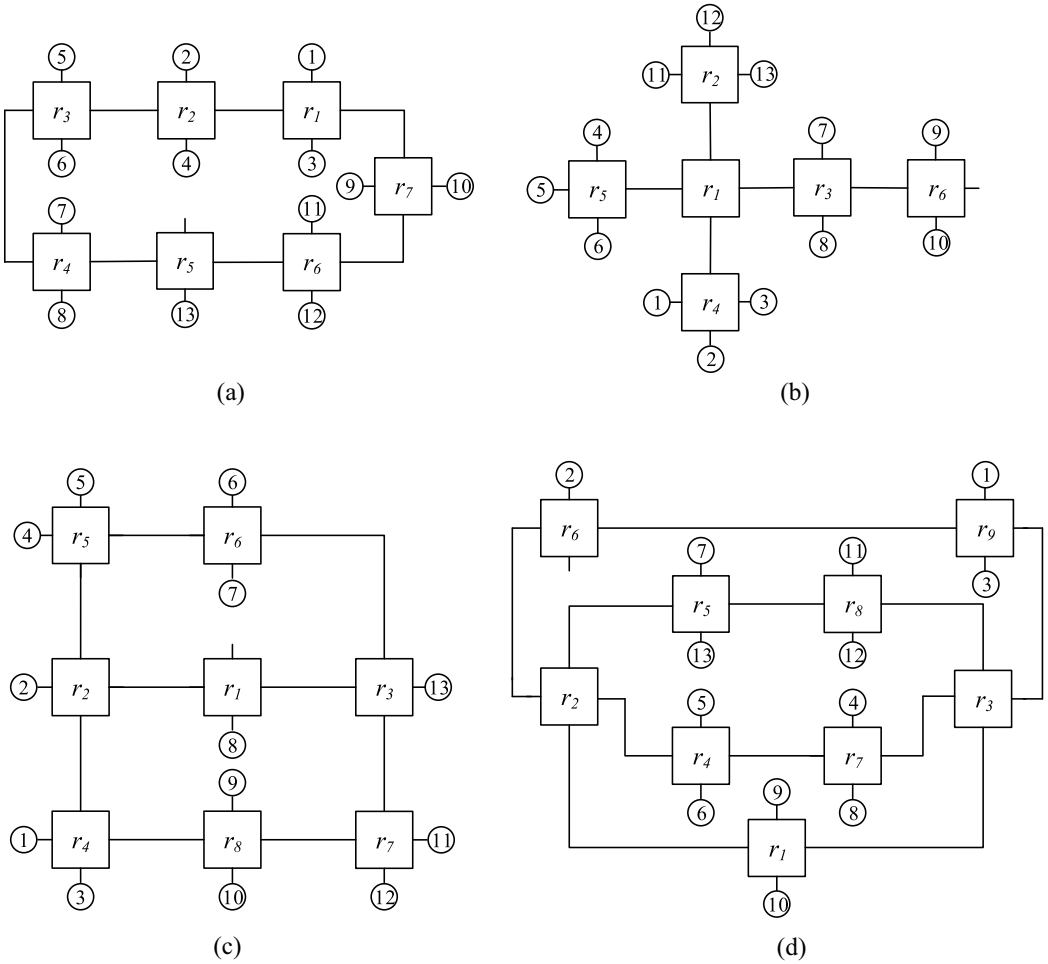


Fig. 9. Mappings of MP3 encoder application onto (a) ring, (b) N-FTTG, and (c) and (d) two FTTGs.

D. Routing

Routing is an important mechanism in NoC designs. For healthy communication in the network, packets must be sent and received on time. To achieve this goal, deadlocks must be avoided and bandwidth requirements fulfilled. In our evaluations above, we used the shortest path routing as the default routing. This routing is static and saved in the routing tables (RTs) of each router. However, when there is a link failure on the default routing path, we cannot use the chip. Therefore, there must be an alternative RT to cover a link failure. As a consequence, we should have more than one RT to cover all single link failures. The default routing ensures the latency and bandwidth constraints and is optimized for energy consumption, but alternative routings may not guarantee these constraints and the energy consumption may not be optimal. Alternative RTs can be powered up by the chip's external pins. For example, with two external pins, we can have four different static RTs for the application.

If we create an RT for each link failure, we end up with $|L|$ RTs, which needs $\lceil \log_2 |L| \rceil$ external pins for the chip to select the required RT. However, each RT may cover more than one link failure since the topology may have more than one extra links. A topology with r routers needs as least $r-1$ links to be fully connected. For example, the topology in Fig. 9(c) has eight routers and nine links, which means that two extra links (el) can be removed from the topology and the topology can still be fully connected. In short, we can remove $el = l - r - 1$ extra links from an FTTG topology without disconnecting it. When we select the links to be deleted, we should make sure that the remaining is a fully connected network and energy increase of the alternative RT is minimum. After determining which links will be covered by the alternative RT, we can use the shortest path routing algorithm to generate the RTs.

VIII. Conclusion

In this paper, we present a fault-tolerant application-specific topology-generation algorithm and an SA-based mapping algorithm. Our FTTG algorithm generates topologies such that each router of the topology can be reached from any router with at least two alternative paths. The generated topology can be used to tolerate at least one link failure by applying the packets' alternative routings. We compare our method with nonfault-tolerant topologies and show that with only a small increase in area, our method brings fault tolerance capability to NoC designs.

References

[1] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Design Autom. Electron. Syst.*, vol. 12, no. 3, Aug. 2007, Art. ID 23.

[2] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. IEEE Design Autom. Conf.*, Las Vegas, NV, USA, 2001, pp. 684–689.
[3] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[4] S. Tosun, Y. Ar, and S. Ozdemir, "Application-specific topology generation algorithms for network-on-chip design," *IET Comput. Digit. Tech.*, vol. 6, no. 5, pp. 318–333, Sep. 2012.
[5] S. Tosun, "New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs," *J. Syst. Archit.*, vol. 57, no. 1, pp. 69–78, 2011.
[6] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proc. Int. Symp. Low Power Electron. Design*, San Diego, CA, USA, 2005, pp. 387–392.
[7] S. Tosun, "Cluster-based application mapping method for network-onchip," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 868–874, 2011.
[8] K.-C. Chang and T.-F. Chen, "Low-power algorithm for automatic topology generation for application-specific networks on chips," *IET Comput. Digit. Tech.*, vol. 2, no. 3, pp. 239–249, May 2008.
[9] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming based techniques for synthesis of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 407–420, Apr. 2006.
[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[11] G. Leary, K. Srinivasan, K. Mehta, and K. S. Chatha, "Design of network-on-chip architectures with a genetic algorithm-based technique," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 5, pp. 674–687, May 2009.
[12] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *Proc. Int. Workshop Parallel Comput. Rout. Commun. (PCRCW)*, Seattle, WA, USA, 1994, pp. 241–255.
[13] K. Constantinides *et al.*, "BulletProof: A defect-tolerant CMP switch architecture," in *Proc. 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, 2006, pp. 5–16.
[14] S.-J. Pan and K.-T. Cheng, "A framework for system reliability analysis considering both system error tolerance and component test quality," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Nice, France, 2007, pp. 1–6.
[15] D. Fick *et al.*, "A highly resilient routing algorithm for fault-tolerant NoCs," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Nice, France, 2009, pp. 21–26.
[16] C. J. Glass and L. M. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 6, pp. 620–636, Jun. 1996.
[17] M. E. Gomez *et al.*, "An efficient fault-tolerant routing methodology for meshes and tori," *IEEE Comput. Archit. Lett.*, vol. 3, no. 1, p. 3, Jan./Dec. 2004.
[18] C.-T. Ho and L. Stockmeyer, "A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers," *IEEE Trans. Comput.*, vol. 53, no. 4, pp. 427–438, Apr. 2004.
[19] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," *ACM SIGARCH Comput. Archit. News*, vol. 32, no. 2, p. 198, Mar. 2004.
[20] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient unicast and multicast support for CMPs," in *Proc. Int. Symp. Microarchit. (MICRO)*, Lake Como, Italy, 2008, pp. 364–375.
[21] J. Wu, "A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1154–1169, Sep. 2003.
[22] J. Zhou and F. C. M. Lau, "Multi-phase minimal fault-tolerant wormhole routing in meshes," *Parallel Comput.*, vol. 30, no. 3, pp. 423–442, Mar. 2004.
[23] E. Cannella *et al.*, "Towards an ESL design framework for adaptive and fault-tolerant MPSoCs: MADNESS or not?" in *Proc. 9th IEEE Symp. Embedded Syst. Real-Time Multimedia (ESTIMedia)*, Taipei, Taiwan, Oct. 2011, pp. 120–129.
[24] E. Cannella, O. Derin, P. Meloni, G. Tuveri, and T. Stefanov, "Adaptivity support for MPSoCs based on process migration in polyhedral process networks," *VLSI Design*, vol. 2, Jan. 2012, Art. ID 987209.
[25] M. Palesi, S. Kumar, and V. Catania, "Leveraging partially faulty links usage for enhancing yield and performance in networks-on-chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 3, pp. 426–440, Mar. 2010.
[26] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos, "A dynamically adjusting gracefully degrading link-level fault-tolerant mechanism for NoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 8, pp. 1235–1248, Aug. 2012.
[27] Y. F. Teh, Z. Qian, and C.-Y. Tsui, "A fault-tolerant NoC using combined link sharing and partial fault link utilization scheme," in *Proc. IEEE/IFIP 19th Int. Conf. VLSI Syst. Chip (VLSI-SoC)*, Hong Kong, Oct. 2011, pp. 296–301.
[28] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "ARIADNE: Agnostic reconfiguration in a disconnected network environment," in *Proc. Int. Conf. Parallel Archit. Compil. Tech. (PACT)*, Galveston, TX, USA, Oct. 2011, pp. 298–309.
[29] D. Fick *et al.*, "Vicis: A reliable network for unreliable silicon," in *Proc. 46th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jul. 2009, pp. 812–817.
[30] D. DiTomaso, A. K. Kodi, and A. Louri, "QORE: A fault tolerant network-on-chip architecture with power-efficient quad-function channel (QFC) buffers," in *Proc. 20th IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Orlando, FL, USA, 2014, pp. 320–331.
[31] R. Parikh and V. Bertacco, "uDIREC: Unified diagnosis and reconfiguration for frugal bypass of NoC faults," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, New York, NY, USA, 2013, pp. 148–159.
[32] S. Tosun, V. B. Ajabshir, O. Mercanoglu, and O. Ozturk, "Fault-tolerant irregular topology design method for network-on-chips," in *Proc. 17th Euromicro Conf. Digit. Syst. Design (DSD)*, Verona, Italy, Aug. 2014, pp. 631–634.
[33] A. Dharwadker and J. Tevet, "The graph isomorphism algorithm," in *Proc. Struct. Semiot. Res. Group*, Tallinn, Estonia, 2009, pp. 1–30.
[34] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972.
[35] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jimenez, "Reducing network-on-chip energy consumption through spatial locality speculation," in *Proc. 5th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, Pittsburgh, PA, USA, May 2011, pp. 233–240.
[36] C. A. M. Marcon, E. I. Moreno, N. L. V. Calazans, and F. G. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," *IET Comput. Digit. Tech.*, vol. 2, no. 6, pp. 471–482, Nov. 2008.
[37] M. Janidarmian, A. Khademzadeh, and M. Tavanpour, "Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile-based network on chip," *IEICE Electron. Exp.*, vol. 6, no. 1, pp. 1–7, Jan. 2009.
[38] (2015, Mar. 24). *International Technology Roadmap for Semiconductors*. [Online]. Available: http://public.itrs.net/

Suleyman Tosun received the M.Sc. and Ph.D. degrees in computer engineering from Syracuse University, Syracuse, NY, USA, in 2001 and 2005, respectively.

He is currently an Associate Professor with the Department of Computer Engineering, Hacettepe University, Ankara, Turkey. His current research interests include electronic design automation, low-power design, network-on-chips, and computer architecture.

Dr. Tosun was a recipient of the Tubitak Career Award. He is a Management Committee Member of EU COST actions IC0805 and IC1204.

Vahid B. Ajabshir received the B.Sc. degree from the University College of Nabi Akram, Tabriz, Iran, in 2008. He is currently pursuing the M.Sc. degree with Ankara University, Ankara, Turkey, both in computer engineering.

His current research interests include network-on-chips, computer vision, radio-frequency identification (RFID), and fuzzy control systems.

Ozge Mercanoglu received the B.Sc. degree in computer engineering from Ankara University, Ankara, Turkey, in 2012, where she is currently pursuing the M.Sc. degree.

Her current research interests include network-on-chips and computer vision.

Ozcan Ozturk received the B.Sc. degree in computer engineering from Bogazici University, Istanbul, Turkey, the M.S. degree in computer engineering from the University of Florida, Gainesville, FL, USA, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, State College, PA, USA, in 2000, 2002, and 2007, respectively.

He is currently an Associate Professor with the Department of Computer Engineering, Bilkent University, Ankara, Turkey. He was with the Cellular and Handheld Group, Intel, Santa Clara, CA, USA, and Marvell, Hamilton, Bermuda. He also held positions with NEC Laboratories, Princeton, NJ, USA, and Arizona State University, Tempe, AZ, USA. His research has been recognized by Fulbright, Portland, OR, USA, Turk Telekom, Ankara, Turkey, IBM, Armonk, NY, USA, Intel, Tubitak, Gebze, Turkey, and EC FP7. His current research interests include manycore architectures, parallel computing, and computer architecture.