

# Routability and Fault Tolerance of FPGA Interconnect Architectures

Jing Huang, Mehdi Baradaran Tahoori, and Fabrizio Lombardi  
Dept of Electrical and Computer Engineering  
Northeastern University  
Boston, MA 02115  
{hj, mtahoori, lombardi}@ece.neu.edu

## Abstract

*This paper presents a new approach for the evaluation of FPGA routing resources in the presence of interconnect faults. All possible interconnect faults for programmable switches and wiring channels are considered. Signal routing in the presence of faulty interconnect resources is analyzed at both switch block and the entire FPGA. Two new probabilistic routing (routability) metrics are proposed and used as figures of merit for evaluating the interconnect resources of commercially available FPGAs as well as academic architectures.*

## I. INTRODUCTION

Field programmable gate arrays (FPGAs) are programmable platforms for many applications such as networking, signal processing, and fault tolerant computing. The programmability of FPGAs has helped to achieve a short design cycle and low development costs, as well as a reduced time-to-market. In an SRAM-based FPGA, all logic elements and programmable switches can be reprogrammed by loading a configuration bitstream, giving the FPGA the flexibility to implement many digital circuits on the same piece of silicon.

An FPGA usually consists of a *two-dimensional (2D)* array of logic resources connected by routing resources. Other cores, such as memory blocks and microprocessors can also be embedded. Over 80% of transistors inside the FPGA are dedicated to the programmable routing network as programmable switches and buffers. Also, more than eight metal layers are used for wiring the interconnects [5]. Hence, FPGAs are very vulnerable to all sorts of interconnect defects, including manufacturing and reliability defects. Although most of these defects are covered by manufacturing test, permanent defects will still affect normal operation of FPGA-based systems due to latent manufacturing flaws (reliability defects) and wearout mechanism. As the level of integration increases aggressively, chips become exponentially more vulnerable to defects during lifetime operation. The correct functionality of the interconnect resources has a significant impact on the design that can be implemented in the FPGA.

The programmability and modularity of an FPGA are readily adaptable to fault-tolerance [6][8]. Adaptive fault-tolerant schemes based on FPGAs use *self repair* techniques to avoid

faulty resources after diagnosing them [7][14][16]. The first step in self repair is to use periodic test or *concurrent error detection (CED)* to detect faults in run time. When faults are detected, the second step is to use diagnosis techniques to locate the faulty resources. Usually, only a small portion of interconnect resources is used, so the third step attempts to remap the design in the FPGA to bypass the faulty resources, such that the desired design can still function correctly.

The availability and flexibility of the interconnect resources often determine whether it is possible to successfully reconfigure the FPGA at possibly a small performance penalty. Here performance penalty refers to rerouting with a longer path and hence longer delay. The structure of the switch block is important for routing and flexibility of reconfiguration. Therefore, fault-tolerance of different FPGA architectures and their resources (in particular the switch block structure) must be evaluated. This work studies the ease and feasibility of remapping in the presence of faulty resources for different switch block architectures. Routability (the probability to route) can not be fully accessed only at switch block level. The entire FPGA array needs to be investigated to obtain information on the routability of the device. Additionally, contemporary FPGAs often have segmented wires that span over multiple switch blocks, such as the Hex lines in the Xilinx Virtex FPGA. These routing resources need to be analyzed at the switch block array level.

In this paper, a detailed study of different FPGA architectures in the presence of various interconnect faults is presented. The impact of faulty interconnect resources on the probability to route (routability) in a switch block array is assessed. This can be used to evaluate interconnect architectures based on different defect rates. Two new metrics are proposed to establish the routability and connectability of the switch block array. The first metric captures the impact of faults on the performance of remapping the design, whereas the second considers the ability to route signals. Moreover, the variation of these metrics for evaluating the capability of different FPGA architectures in *remapping* the design (to avoid faults) is also considered. The impact of different types of faults is analyzed and compared for various FPGA switch block structures.

The presented metrics for routability and connectability are evaluated for commercial and academic FPGAs. The

fault tolerance capabilities of FPGA architectures are also compared. A polynomial time algorithm which finds all paths (up to a maximum length) between a pair of primary I/O endpoints in a switch block array, is presented. This algorithm is used as a basis for analyzing switch blocks as well as switch block arrays based on paths as the criteria for routing and fault tolerance. The proposed method is applicable to arrays of arbitrary switch block structures; for each switch block structure, routability must be executed only once to fully assess its fault tolerance.

The rest of this paper is organized as follows. A brief review of previous work in the technical literature as well as the preliminaries and fault models are given in Section II. In Section III, the proposed routability and unconnectability metrics are presented. Section IV presents the algorithm. The results for commercial and academic FPGAs can be seen in Section V. Finally, Section VI concludes the paper.

## II. BACKGROUND

### A. Previous Work

The programmability of FPGA can readily be used in adaptive fault-tolerant schemes. In [8], a comprehensive approach to reconfigure one-time programmable FPGAs with faulty logic resources has been proposed. Methods for tolerating faults in the FPGA logic resources by shifting the configuration data are presented in [6]. In [14], a column-based reconfiguration technique using precompiled configurations is presented. The ROAR (Reliability Obtained by Adaptive Reconfiguration) project at the Stanford Center for Reliable Computing is an example of reconfigurable architecture with self-repairing ability [16]. ROAR achieves fault tolerance by CED, fast fault location and quick recovery from temporary failures and fast repair of the system from permanent faults. Another self-repairing example is the roving STAR approach proposed in [1]. Testing takes place in the self-testing area (STAR) while the system is running. The entire chip is tested by roving the STAR across the FPGA. Additionally, as explained in Sec. I, testing and diagnosis are integral parts of self repair techniques. These issues have been addressed by many papers. Application-independent testing of routing resources in FPGAs has been addressed in [6][7][12] [15][17][18][19]. Application-dependent testing of FPGAs has been reported in [4][11]. Diagnosis of interconnect faults in FPGAs has been discussed in [9][10][20][21].

However, previous work can not be used to establish and assess the most appropriate switch block structure for fault tolerance. Very little work has been reported under which conditions a certain configuration is realizable in the FPGA when faults are present. This is directly related to the capability of a switch block structure and the overall FPGA to route in the presence of faults. This effectively differentiates the switch block structures currently available from different manufacturers and their ability to provide routing with faulty interconnect resources. In this paper we propose methods to evaluate fault tolerance in arbitrary switch block structures; various switch blocks, including the Xilinx XC4000 and the

Virtex FPGA as well as academic FPGAs, are analyzed in detail and a quantitative evaluation is pursued.

### B. Preliminaries

The FPGA model used in this paper is similar to Xilinx FPGAs[22]. It consists of a two dimensional array of *configurable logic blocks (CLBs)* and switch blocks. Programmable switch blocks provide the selective connectivity of horizontal as well as vertical routing channels placed between the logic blocks which consists of line segments (nets) of different length. Note that these nets may or may not be buffered. Inside each switch block programmable switches can be found; a switch is a pass transistor controllable by a user-programmable SRAM cell. These switches provide selective connectivity between pairs of line segments connected to the switch block. A switch can be *bidirectional*, which means signal can be routed from endpoint *A* to endpoint *B* or endpoint *B* to endpoint *A*; or *uni-directional*, which means signal can only be routed from endpoint *A* to endpoint *B* but not vice versa. The values of these SRAMs are part of the configuration data which is loaded into the chip when the chip is configured. If the switch is closed, the connection between two nets is established; otherwise, these nets are not connected.

In this paper, a generic switch block array composed of arbitrary structured switch blocks is assumed. A 2-D array consists of  $k_r \times k_c$  switch blocks connected in both the horizontal and vertical directions via nets as shown in Figure 1(a). Each switch block has four ports: North, South, West and East; each port has  $n$  endpoints. The primary I/O endpoints of the array are also indicated in the Figure. In this paper we consider paths routed from one primary I/O endpoint to another.

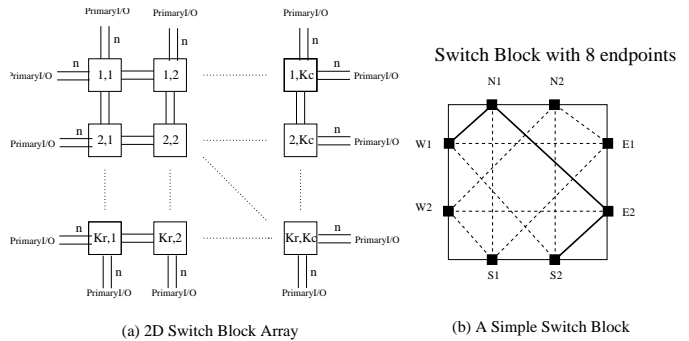


Fig. 1. Switch block and 2-D array of switch blocks

### C. Fault Model

Faults in the FPGA interconnect can be categorized into two major groups, namely *opens* and *shorts*. An open fault can be a switch stuck-open or an open on a net (line segment). A switch *stuck-open* fault causes the switch to be permanently open regardless of the value of the SRAM cell controlling the switch. A short fault can be a switch stuck-closed or a bridging fault between two routing resources. A switch *stuck-closed* fault causes the switch to be permanently closed regardless

of the value of the memory cell controlling the switch. Two different bridging fault types are considered in this paper. In the first type, any two arbitrary nets can be shorted together. In the second type, by considering physical information, only adjacent nets can have a bridging fault. We do not consider stuck-at faults in the interconnect separately, as these faults can be modeled as shorts to power rails.

### III. ROUTABILITY METRICS

The main objective of this work is to assess the routability of the switch block array in the presence of faults; if some switches are programmed “on”, some paths can be established to route signals from the primary input endpoints to the primary output endpoints. Obviously, routability is related to the number of paths which are realizable in a given switch block array. Moreover, the length of these paths determines the routing delay, which is directly related to the performance of the mapped design. When some switches and/or nets become faulty, the faulty elements and possibly some fault-free resources around them can no longer be used for routing signals. As a result routability is lost due to faults. Some previously connectable pairs of endpoints may no longer be routable. In terms of fault tolerance, it is interesting to establish the change in routability when the number of faults increases. In this paper the following metrics are proposed to evaluate the routability of a switch block.

**Metric1 (Routability):** Let the number of paths of length  $l$  between a given source,  $s$ , and destination,  $t$ , be  $N_l$ , then the proposed metric  $M_1(s, t)$  is defined as:

$$M_1(s, t) = \sum_l \frac{N_l}{l} \quad (1)$$

This value is computed for a large enough sample of all pairs of source and destination vertices in the switch block array and the average value is considered, i.e.  $M_1 = E(M_1(s, t))$ .

**Metric2 (Unconnectability):** The metric  $M_2$  is defined as the number of pairs of vertices that can not be connected (routed) by any path of length  $L_{max}$  or less.

The length of the path is defined as the number of switches and nets in the path. The first metric captures the performance of routability because paths with shorter lengths (i.e. faster paths) have more weight (note that the path length appears in denominator). From a performance point of view, short paths should be favored over long paths. Higher  $M_1$  represents more routable switch block arrays with higher performance. Obviously, the larger the number of paths, the more routable and flexible the switch block array is. The number of paths is dependent on the number of switches as well as their distribution in the switch blocks. Therefore, the proposed metric correctly considers both the number of paths and their lengths. The presence of faulty elements reduces the number of paths, because it does not allow a signal to be routed within a certain path length.

The second metric models two situations. In the first situation by setting  $L_{max}$  to infinity (a large enough value in

practice), it evaluates the number of pairs which cannot be routed in the presence of interconnect faults. In the second situation, by setting  $L_{max}$  to a predefined threshold, this metric models the number of pairs which cannot be routed within a given length. For some performance critical applications, it is extremely important that the routing delay does not exceed some predefined threshold. It is also important to evaluate such metric for the pairs in the critical paths.

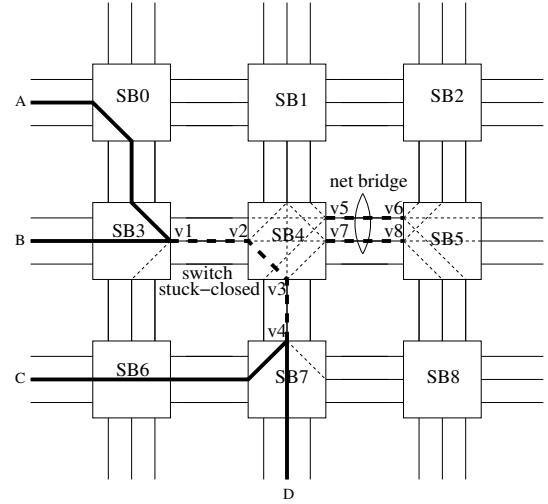


Fig. 2. Short faults in  $3 \times 3$  array of switch blocks

Different types of faults have different impacts on routability, since the type of the fault determines which resources become unusable. In the presence of open faults, the faulty element (switch or net) is permanently disconnected, hence it can no longer be used; other interconnect resources are not affected by open faults and can still be used. Therefore, only one switch block is sufficient to study the behavior of interconnect in the presence of switch stuck-open faults.

However, a short fault not only makes shorted resources unusable, but it also affects the utilization of some resources in the neighboring switch blocks. An example is shown in Figure 2. A switch  $v2$ - $v3$  stuck-closed fault is present in the center switch block (SB4). As  $v2$  and  $v3$  are shorted together, then the nets connecting these two vertices are also shorted (connected) together, i.e. the net  $v1$ - $v2$  is connected to net  $v3$ - $v4$  (this is represented by the highlighted dotted line in Figure 2). Now consider the routed path from  $A$  to  $B$ , and the routed path from  $C$  to  $D$ , shown as highlighted solid lines. Although these two paths do not utilize the faulty switch  $v2$ - $v3$ , they are still shorted together due to this fault. Therefore, to ensure correct routing, the four vertices  $v1, v2, v3$  and  $v4$  along with all the nets and switches connected to these vertices must be marked as “unusable”, i.e. they can no longer be used for rerouting signals. This example clearly shows how some fault-free resources in the neighboring blocks are affected by switch stuck-closed faults.

A similar scenario occurs for net bridging faults. In the example shown in Figure 2, if a bridging fault is present between net  $v5$ - $v6$  and net  $v7$ - $v8$ , the four vertices ( $v5, v6, v7$

and  $v8$ ) along with all the switches connected to them can no longer be used for rerouting. In summary, a short fault has a bigger impact on routability than an open fault.

Based on the above observation, the smallest array for studying the impact of short faults is a  $3 \times 3$  array of 9 switch blocks, in which all neighbors of the central switch block must be considered. The  $3 \times 3$  array is used for both open and short faults in the central switch block. An example of this array is shown in Figure 2. Segmented wires can also be included. In general at least a  $k \times k$  switch block array is needed to study the segmented wire that span  $k$  switch blocks.

#### IV. ROUTABILITY EVALUATION METHOD

##### A. The Path-based Method

In this section, a path-based method is presented to evaluate the fault tolerance of a switch block array using the metrics  $M_1$  and  $M_2$  defined in Sec. III. The switch block array is modeled as an unweighted, undirected graph  $G = (V, E)$ ; each endpoint of the switch block array is a vertex in  $G$ ; if a programmable switch or a net exists between two endpoints, an edge is added in  $E$  between these two vertices. Let the minimal degree of the vertices in  $G$  be  $d_{min}=d$ .  $|E| = e$  denotes the number of switches and nets in the switch block array. Figure 3 shows an example of a simple  $3 \times 3$  switch block array.

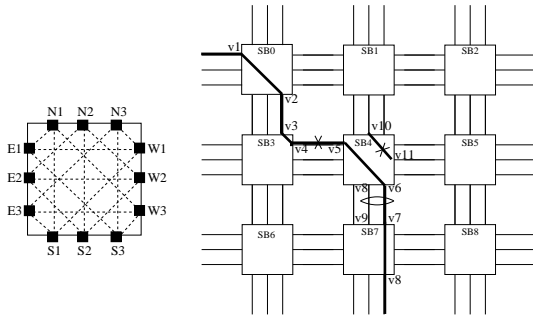


Fig. 3. An array example with a path of length 7

Here examples are given to demonstrate the effect of faults on the routed paths. In Figure 3, a path from an array primary input endpoint  $v1$  to a primary output endpoint  $v8$  of length 7 is highlighted. This path is denoted as  $v1-v2-v3-v4-v5-v6-v7-v8$ . If there is a net open fault on the net  $v4-v5$ , or a net bridging fault between the net  $v6-v7$  and the net  $v8-v9$ , this path can not be realized. However, some other faults (such as switch stuck-open fault in switch  $v10-v11$ ) have no effect on this path. Since the objective is to evaluate the fault tolerance of different interconnection structures, the presented metrics must be evaluated for both fault-free and faulty cases. Long paths are not desirable because of unacceptable routing delay, therefore the longer paths have a smaller contribution to  $M1$ . In our experiments we limit the path length to a predefined  $l_{max} = 10$ .

The evaluation method can be described as follows. Since the number of all possible (source, destination) pairs are quadratic to the size of the graph, it becomes intractable

to perform the above procedure for all pairs. Therefore, a statistical study can be conducted by considering a large enough sample of randomly generated pairs.  $N_p$  randomly generated (source,destination) pairs are chosen first, then all paths of length up to  $l_{max}$  between each pair of vertices are found. Let the number of paths of length  $l$  be  $N_l$ . The first metric is calculated as follows:

$$M_1 = AVG_{N_p \text{ pairs}} \left( \sum_l \frac{N_l}{l} \right) \quad (2)$$

If for a specific pair no path is found, then this pair is considered as unconnectable. Metric  $M_2$  counts the number of unconnectable pair out of the  $N_p$  randomly generated (source, destination) pairs. Then, faults are injected into the array; the injected faults can be any of the four types in the previously described fault model. Assume  $f$  faults are injected. We consider only  $N_f$  randomly generated fault patterns with  $f$  faults, because the number of all possible fault patterns is exponential to the size of the switch block array.  $M_1$  is calculated based on an average over these  $N_f$  fault patterns. The above process is performed for  $N_p$  randomly generated pairs of vertices to obtain the average metric over the  $N_p$  pairs. This is denoted by  $M_1^{(f)}$ , i.e. the routability in presence of  $f$  faults. We also count the unconnectable pairs in the presence of  $f$  faults to obtain  $M_2^{(f)}$ .

To evaluate the fault tolerance of a certain array we inject  $f = 1, 2, 3, \dots$  faults to obtain  $M_1^{(f)}$  and  $M_2^{(f)}$ . This can be considered as the fault rate, or the number of faults over time. The injected faults in each set of experiment can be of the same or different types.

Contemporary FPGAs have not only single lines (nets) connecting adjacent switch blocks but also segmented lines that span over multiple switch blocks. For example the Xilinx XC4000 switch block has *double lines* that run past two CLBs before entering a switch block. Also the Virtex switch block has *Hex Lines* to route signals to another switch block six blocks away. Similarly, a *Long Line* connects switch blocks which are twelve tiles apart. Since the switch block array is modeled as a graph, the proposed method is applicable to switch blocks with segmented interconnect as well, the segmented nets are also modeled as edges in the graph. Experimental results of Virtex array with Hex lines are shown in the result section.

##### B. The Algorithm

In this section, the algorithm for finding all paths up to length  $l_{max}$  between two endpoints in an arbitrary graph is presented. As described in Sec. III, only a  $3 \times 3$  switch block array is needed. Here,  $l_{max}$  is set to 10. Note that this algorithm is applicable to arrays of switch blocks of arbitrary structures. This algorithm is based on a Depth First Search [3] and finds all possible paths of length up to  $l_{max}$  in an unweighted graph  $G = (V, E)$ , starting from a vertex  $s \in V$  to a destination vertex  $t \in V$ . Let the minimal degree of the graph (i.e. for all vertices in  $G$ ) be  $d_{min}$  and the maximum degree be  $d_{max}$ .

The graph  $G$  is stored as an adjacency list.  $s$  is the source vertex,  $t$  is the destination vertex,  $len$  is the maximum length allowed for the desired path.  $Path$  contains the current path, the length of the current path is given by  $k$ .

Function ListPath( $s,t,len$ ):

input:source vertex  $s$   
target vertex  $t$   
length of the path required  $len$

1. **if** this is top call **then**
2.      $Path \leftarrow \emptyset, k \leftarrow 0$ ;
3.      $Path \leftarrow Path \cup \{s\}$
4.  $s \leftarrow$  the  $k^{th}$  element of  $Path$
5. **if**  $s=t$  **then**
6.     store  $Path$ ;
7.     **return**;
8. **else if**  $k=len$  **then**
9.     **return**;
10.  $u \leftarrow$  first vertex in AdjList( $s$ );
11. **while** still more vertex in AdjList( $s$ ) **loop**
12.     **if**  $u \in Path$  **AND**  $u \neq t$  **then**
13.         **continue**
14.      $Path \leftarrow Path \cup \{u\}, k \leftarrow k+1$
15.     **call** listPath ( $s,t,len$ )
16.      $k \leftarrow k-1$
17.      $Path \leftarrow Path - \{u\}$
18.      $u \leftarrow$  next vertex in AdjList( $s$ )

This recursive algorithm starts with the source  $s$  and searches the tree in a depth-first manner. In the top call,  $Path$  is initialized to contain only  $s$ , and  $k$  is set to 0 (lines 1-3). In every subsequent call,  $s$  is set to be the last vertex of the already established path (line 4), whose length is  $k$ . Next, if a path with an acceptable length (less than  $l_{max}$ ) is found, the path is stored and the function returns. Otherwise, if the path has the maximum length but hasn't reached  $t$ , this path is discarded (lines 5-9). If the path hasn't reached its maximum length and hasn't reached  $t$ , we search deeper by going to the next unsearched neighbor of  $s$  and call the algorithm recursively to find all the desired paths. Applying this algorithm to a switch block array is straightforward as the nets connecting neighboring switch blocks are also considered as edges in the graph.

The application of the above algorithm on the switch block shown in Figure 1(b), for  $l_{max}=3$  is given. The resulting output for the vertex pair ( $W1,E1$ ) is:

- Paths of Length 1:  $W1 - E1$ .
- Paths of Length 2: none.
- Paths of Length 3:  $W1 - N1 - S1 - E1$ ,  $W1 - S2 - N2 - E1$ .

The above algorithm is recursive and its complexity is computed as follows. Since the maximum degree of the vertices in  $G$  is at most  $n-1$ , the while loop (line 12) executes at most  $n-1$  times. The recursive depth is at most  $l_{max}$ , where

$l_{max}$  is the maximum path length. If the algorithm is executed to find all paths of length up to  $l_{max}$  between source and destination in a fully connected graph, the time complexity is  $O(n^{l_{max}})$ , where  $l_{max}$  is a constant for this particular problem.

It is an enumeration problem because all paths of length up to  $l_{max}$  (starting from a source vertex in an arbitrary graph  $G$ ) must be found. So this problem belongs to the category of *polynomial delay* complexity [13], as the consecutive solutions of the enumeration problem can be produced with polynomial time delay. However, switch blocks are sparsely connected (i.e.  $d_{max} \ll n$ ), so its complexity is  $O(d_{max}^{l_{max}})$ , which in practice is considerably smaller than  $O(n^{l_{max}})$ . Many FPGAs have  $d_{min} = d_{max} = d = 3$  inside the switch block (or  $d_{max} = 4$  when considering the edge establishing connection between switch blocks). In this case the complexity is  $O(4^{l_{max}})$ . Note that an additional step in  $O(|E|)$  time is required to read the graph and build the adjacency list.

## V. RESULTS

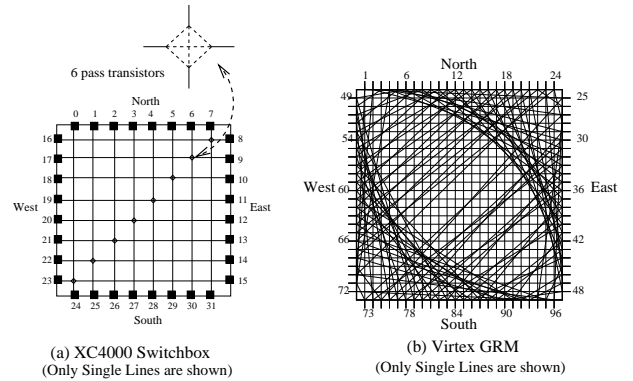


Fig. 4. The XC4000 and Virtex Switch Block

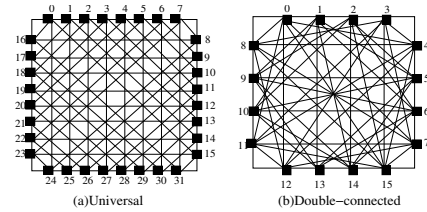


Fig. 5. Universal Switch Block and Double-Connected Switch Block

In this section, the routability metrics are computed for some commercial FPGA architectures, namely the Xilinx XC4000 and Virtex FPGAs, as well as some academic FPGAs [2]. In particular, the following FPGA architectures are investigated in this work

- **XC4000:** This switch block structure is shown in Figure 4(a) [22]. It has 8 endpoints in each of the four ports, i.e. a total of  $8 \times 4 = 32$  endpoints. Each endpoint has degree  $d = 3$ , so altogether there are 48 switches in the switch block.
- **Virtex (Single Lines Only):** The Virtex GRM (General Routing Matrix) is shown in 4(b) [22]. Virtex GRM

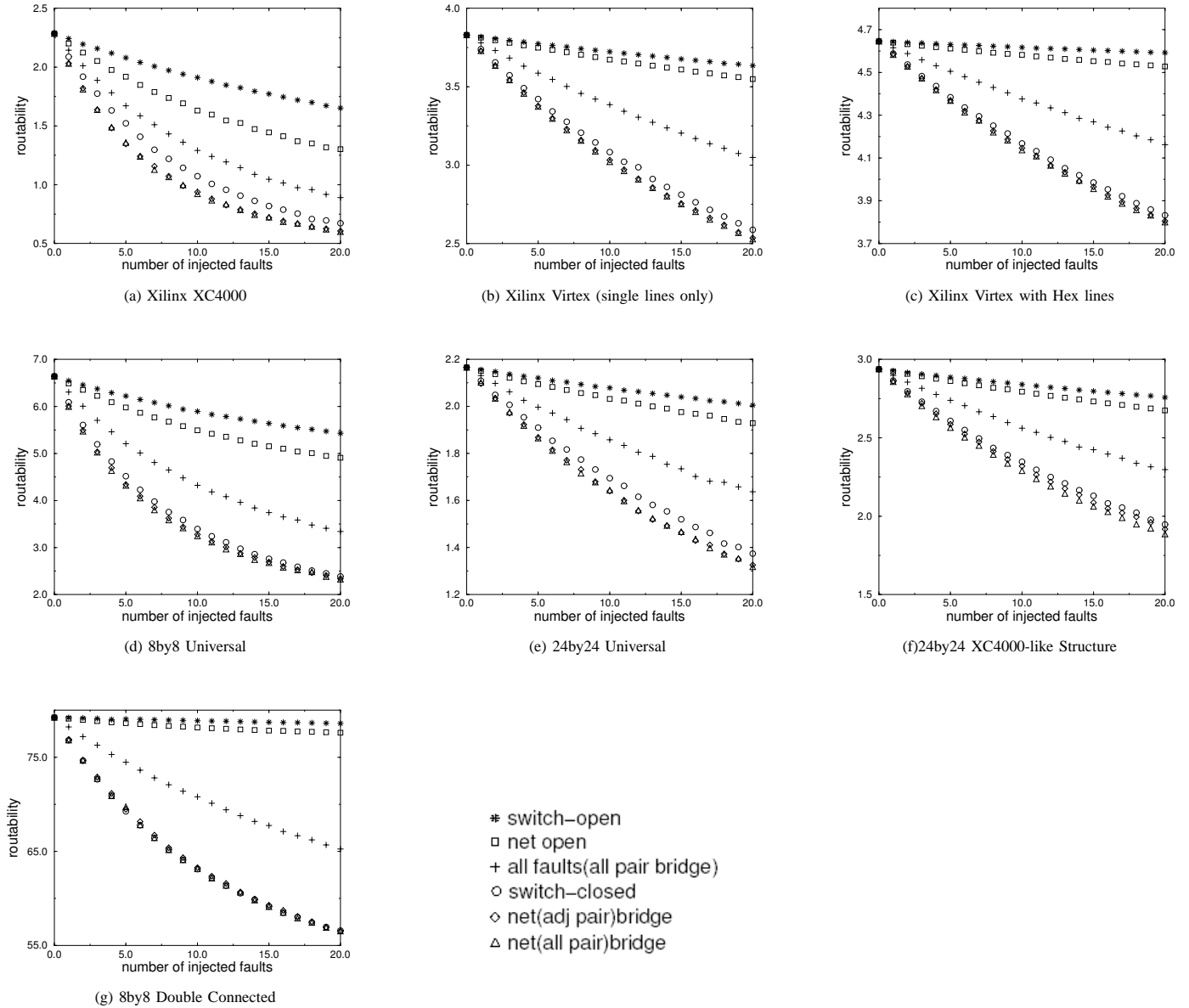


Fig. 7. Routability for Various Structured Switch Block Arrays [Different figures have different Y-axis ranges]

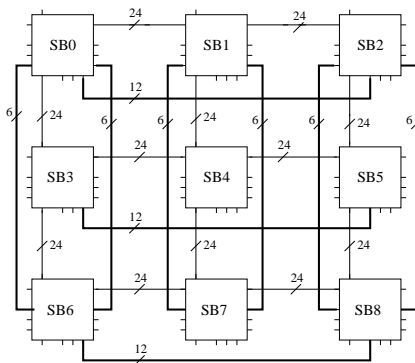


Fig. 6. Array of the Virtex GRM with Hex lines

has 24 single line endpoints in each of the four ports, therefore a total of 96 endpoints. Each endpoint has degree  $d = 3$ , thus the total number of switches is 144 (considering only the switches connecting *single lines* in Virtex architecture).

- **Virtex (with Hex Lines):** We also included the model of the Xilinx Virtex GRM with both single lines and Hex lines connections. Besides the single lines, the Virtex GRM also has 12 horizontal Hex line and 12 vertical Hex lines routing signals to another switch block 6 blocks away. However since we only consider a  $3 \times 3$  array in this paper, we assume that the Hex lines connects the switch blocks in column (row) 1 to switch block in

column (row) 3. The  $3 \times 3$  Virtex array with Hex lines are shown in Figure 6, where the Hex lines are represented by highlighted lines. Note that some of the Hex-Hex, Hex-Single switches in this switch block are uni-directional. All Single-Single switches in all the different types of switch blocks considered in this paper are bidirectional.

- **Universal:** This universal structure has been proposed in [2]. An  $8 \times 8$  universal switch block (32 endpoints) is shown in Figure 5(a). The degree of each endpoint is three (similar to XC4000 and Virtex). This switch block has 48 switches. To compare this structure with the Virtex switch block structure, a  $24 \times 24$  universal switch block with 96 endpoints and 144 switches, is also considered.
- **Double-Connected:** To further investigate the effect of connectivity on the fault tolerance of the switch block, a *double-connected* universal switch block is considered in which each endpoint has a degree of 6. As an example, a  $4 \times 4$  double-connected switch block with 48 switches is shown in Figure 5(b); each endpoint is connected to two endpoints in each other port.

In simulation runs, 1500 (source, destination) pairs were randomly chosen (i.e.  $N_p = 1500$ ), and 100 fault patterns were randomly injected for each faulty situation ( $N_f = 100$ ). The results are computed based on an average over these fault patterns in each case.

All types of faults based on the fault models presented in Sec. II.C are injected, namely switch stuck-open, switch stuck-closed, net open, and net bridging (adjacent coupling and random coupling) faults. Figures 7 and 8 show the results of routability and unconnectability for each architecture based on the different types of faults, respectively.

The results show that in all architectures, the most severe fault is net bridging fault, while the least severe type is switch stuck-open. The following is the order of faults based on their impact on routability from the least to the most severe: switch stuck-open, net open, switch stuck-closed, net bridging (adjacent coupling), and net bridging (random coupling). This ordering is consistent over all architectures. Hence, open faults have far less impact on routability compared to short faults. Moreover, adjacent and random coupling bridging faults are almost indistinguishable in terms of routability.

The results for unconnectability are presented in Figure 8. As shown in these diagrams, the impact of switch stuck-open faults on connectability is almost negligible, whereas random coupling net bridging faults have the most severe effect. Unlike routability in which random and adjacent coupling have an indistinguishable impact, random coupling may or may not look more severe than adjacent coupling in terms of connectability depending on the architecture.

Figure 9 show the comparison of routability of different architectures based on different fault models. As the routability of the  $8 \times 8$  double-connected architecture is much higher than the others, it is not included in these figures. The ordering of these architectures is consistent for all faults. As shown in these diagrams, the relative ordering of routability of these architectures in descending order is:  $8 \times 8$  double-connected,

$8 \times 8$  Universal, Virtex with Hex lines, Virtex with only single lines,  $24 \times 24$  XC4000-like,  $24 \times 24$  Universal, and XC4000.

Similar comparison has been considered for unconnectability. Again, the relative ordering of these architectures is consistent for all fault models. The relative connectability of these architectures in descending order (unconnectability in ascending order) is as follows: Virtex with only single lines,  $8 \times 8$  double-connected, Virtex with Hex lines,  $24 \times 24$  XC4000-like,  $8 \times 8$  Universal, XC4000, and  $24 \times 24$  Universal.

Clearly, routability and connectability do not follow the same ordering. For example, routability of the  $8 \times 8$  double-connected switch block is much higher than others, but its connectability is even lower than Virtex's. This means that some architectures offer higher performance in terms of routability in the presence of faults, while others provide more connectable pairs in the presence of faults. So, connectability and high performance routing cannot be achieved at the same time using these architectures. An FPGA user must consider this trade off based on the particular application and specific fault tolerance requirements.

To further investigate the fault-tolerant capabilities of these architectures in terms of ease of remapping a design to avoid faults, the following variations of routability and connectability have also been considered and evaluated:

- **Connectable Routability:** The routability (metric 1) was calculated based on an average over all randomly generated pairs, regardless of the connectability of some of these pairs. Alternatively, the average can be calculated only for those pairs which are connectable in the fault-free switch block. So instead of using Equation 2 for routability, the following equation is used:

$$M'_1 = AVG_{connectable\ pairs} \left( \sum_l \frac{N_l}{l} \right) \quad (3)$$

For example, in the presence of 5 switch stuck-closed faults in the XC4000, instead of computing the average for all 1500 pairs, routability is calculated for only 185 connectable pairs. This metric shows the performance of routing connectable pairs in the presence of faults. The results are shown in Figure 10.

- **Connectability Rate:** The connectability metric (M2) shows the number of connectable pairs in the presence of faults. These figures are biased due to the fault-free unconnectability values. For fault-tolerance, it is also interesting to consider the slope of these curves; i.e. the rate by which unconnectability grows with an increase in the number of faults. Although some architectures offer higher connectability under fault-free conditions (i.e. to provide more flexibility in *mapping* the design), this ability may drop rapidly in the presence of faults. Hence, it may be harder to *remap* the design to avoid faults. Once the design is successfully mapped to the FPGA, the slope of the connectability curve (i.e. *connectability rate*), shows the ease of remapping the design in the presence of interconnect faults. The results for unconnectability rates are shown in Table I. Smaller unconnectability rates

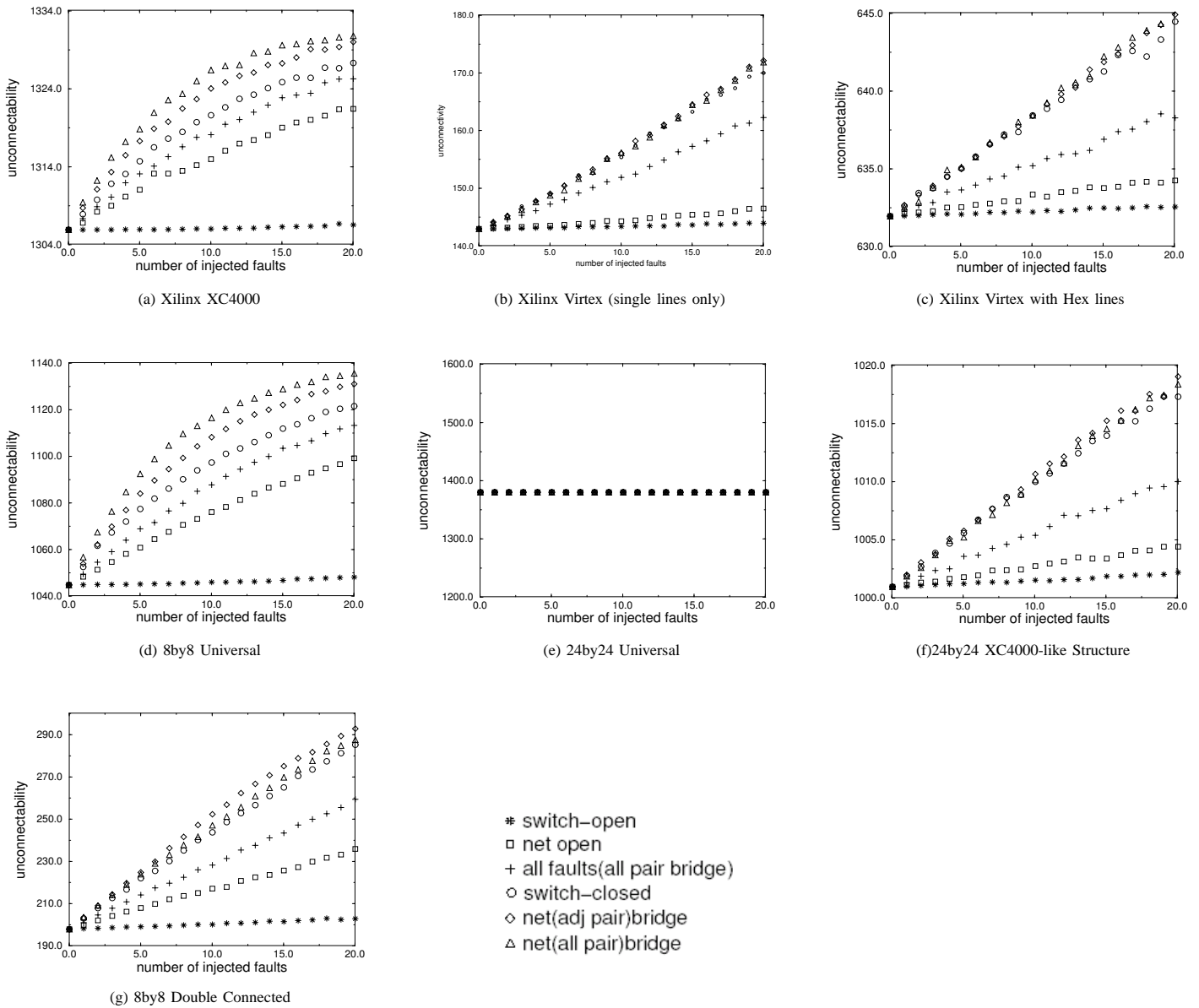


Fig. 8. Unconnectability for Various Structured Switch Block Arrays [Different figures have different Y-axis ranges]

indicate higher fault-tolerance. Unlike the previous cases, the relative ordering of architectures is not the same for all types of faults. Among these architectures, the number of unconnectable pairs doesn't increase with the number of injected faults for the  $24 \times 24$  Universal (although this architecture does not offer high connectability for the fault-free conditions). Also, the  $8 \times 8$  double-connected architecture, which offers the best routability, has the highest unconnectability rate.

### VI. CONCLUSIONS

Defect and Fault tolerant schemes based on FPGA devices, such as adaptive self-repair, are very dependent on the routability of FPGA interconnect architecture. In this paper, a detailed

	Switch Open	Switch Closed	Net Open	Net Bridge	Random Faults
<i>XC4000</i>	0.031	1.071	0.776	1.245	0.969
<i>Virtex</i>	0.05	1.353	0.176	1.446	0.727
<i>8by8Universal</i>	0.196	3.842	2.723	4.542	3.426
<i>24by24Universal</i>	0	0	0	0	0
<i>24by24XC-like</i>	0.062	0.818	0.173	0.871	0.453
<i>8by8Double</i>	0.249	4.37	1.904	4.49	3.078
<i>Virtex(with Hex)</i>	0.030	0.63	0.11	0.662	0.316

TABLE I  
UNCONNECTABILITY RATE OF DIFFERENT SWITCH BLOCK STRUCTURES AND FAULT TYPES

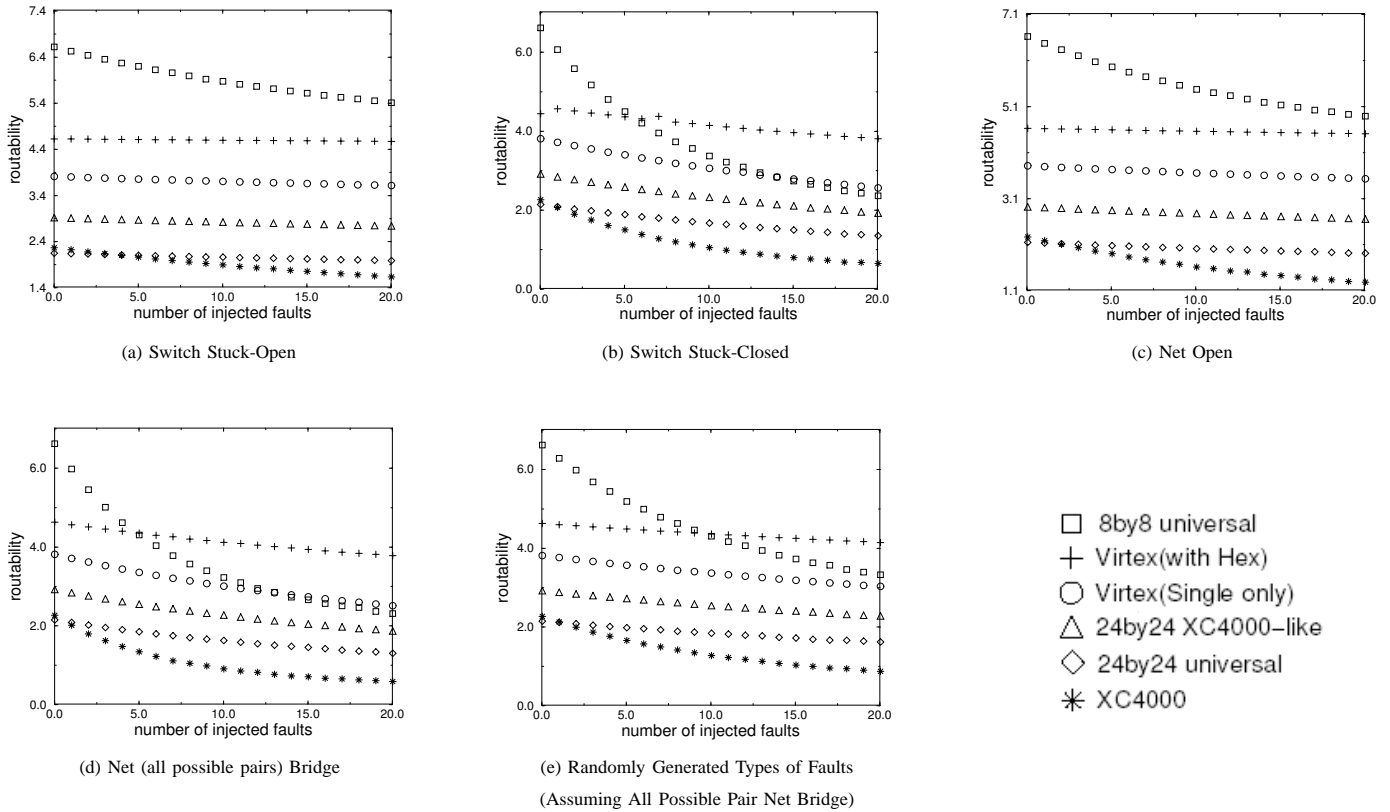


Fig. 9. Routability for Various Types of faults [Different figures have different Y-axis ranges]

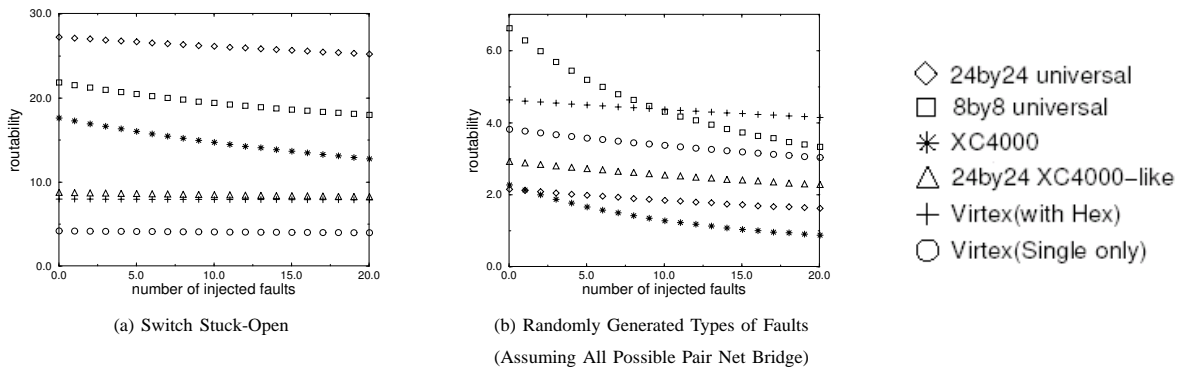


Fig. 10. Routability (connected pairs only) for Various Types of faults [Different figures have different Y-axis ranges]

study of FPGA interconnect architecture in the presence of interconnect faults is presented. Two new metrics, routability and connectability, for evaluating fault tolerance of a switch block array are presented. The impact of different types of interconnect faults, including open and short faults in both nets and programmable switches, on the probability to route (routability) in a switch block array is investigated. These metrics not only consider the connectability (i.e. the number of connectable pairs in the presence of interconnect faults) but they also take the performance of routing into account. Other

variations of these metrics which consider routability and connectability in *remapping* a design to avoid faulty elements, are also considered.

The proposed method is applicable to any switch-based FPGA interconnection architecture; for each switch block structure, routability must be evaluated only once to fully assess its fault tolerance.

This method has been used for evaluating different interconnect architectures in commercial FPGAs (such as the Xilinx XC4000 and Virtex) as well as academic FPGAs. It

has been shown that short faults have more severe impact on routability and connectability than open faults. It is also shown that connectability doesn't guarantee routability at the same time. Hence, some interconnect architectures which offer the maximum number of routable pairs, do not result in more alternative paths with shorter lengths (higher performance) to achieve high routability for the connectable pairs. For some architectures which provide very high routability, their connectability reduces very quickly when increasing the number of faults. The distribution of the switches inside the switch block plays a key role in routability and connectability. Our simulation results show that different architectures with the same number of switches have different routability and connectability. There is a trade-off between routability and connectability which is determined by the switch block structure.

Also, it is shown that routability and connectability of a particular architecture in mapping a design are not the same for remapping that design. Variations of the two proposed metrics are used in order to assess the ease of remapping for self-repair applications. Our experiments show that the relative *connectability rate* of different architectures, as a connectability metric for remapping, is very sensitive to the type of fault.

The metrics provided and evaluated in this paper consider different aspects of fault tolerance and routability. The choice of the most appropriate architecture and the evaluation metric depends on the particular application and fault tolerance requirements of the system.

#### REFERENCES

- [1] M.Abramovici, C.Stroud, C.Hamilton, C.Wijesuriya, V.Verma, "Using Roving Stars for Online Testing and Diagnosis of FPGAs", *Proc. International Test Conference*, pp 973-982, 1999.
- [2] Y.W. Chang, D.F. Wong, C.K. Wong, "Universal Switch-Module Design for Symmetric-Array-Based FPGAs", *Proc. ACM International Symposium on Field Programmable Gate Arrays*, pp. 80-86, 1996.
- [3] T.H.Cormen, C.E.Leiserson,R.L.Rivest, C.Stein "Introduction to Algorithms", *McGraw-Hill* , pp. 643-693, 2001.
- [4] D. Das, N. A. Touba, "A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems", *Proc. International Conference On VLSI Design*, 1999.
- [5] A. DeHon, "Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization)", *Proc. ACM Int'l Symp. On Field Programmable Gate Arrays*, Pages: 69 - 78, 1999.
- [6] A.Doumar, S.Kaneko, H.Ito,"Defect and Fault Tolerance FPGAs by Shifting the Configuration Data", *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI systems*, pp. 377-384, 1999.
- [7] W. Feng, F.J.Meyer, F. Lombardi, "Testing Programmable Interconnect Systems: The General Case.", Internal Report, 2003.
- [8] W.Feng, X.Chen, F.J.Meyer, F.Lombardi,"Reconfiguration of One-Time Programmable FPGAs with Faulty Logic Resources", *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI systems*, pp. 368-376, 1999.
- [9] I.Harris, R.Tessier, "Diagnosis of Interconnect Faults in Cluster-Based FPGA Architectures", *Proc. Design Automation Conference*, pp. 49-54,2000.
- [10] W.Huang, X.Cheng, F.Lombardi, "On the Diagnosis of Programmable Interconnect Systems: Theory and Application", *Proc. IEEE VLSI Test Symposium*, pp. 204-209, 1996.
- [11] W.J.Huang, E.J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance", *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.
- [12] T.Liu, F.Lombardi, J.Salinas, "Diagnosis of Interconnects and FPICs Using a Structured Walking-1 Approach", *Proc. VLSI Test Symposium*, pp 256-261, 1995.
- [13] D.S.Johnson, M.Yannakakis, C.H.Papadimitriou, "On Generating All Maximal Independent Sets", *Information Processing Letters*, 27(3) pp. 119-123, 1988.
- [14] W.-J.Huang, E.J.McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance", *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.
- [15] H.Michinishi, T.Yokohira, T.Okamoto, "A Test Methodology for Interconnect Structures of LUT-Based FPGAs", *Proc. Asian Test Symposium*, pp 68-74, 1996.
- [16] S.Mitra, W.J. Huang, N.Saxena, S.Y.Yu, E.J.McCluskey, "Dependable Reconfigurable Computing: Reliability Obtained by Adaptive Reconfiguration", *ACM Trans. Embedded Computing Systems*, to appear.
- [17] M.Renovell, J.M.Portal, J.Figuras, Y. Zorian, "Testing the Interconnect of RAM-Based FPGAs", *IEEE Design and Test of Computers*, pp. 45-50,1998.
- [18] C.Stroud, S.Wijesuriya, C.Hamilton, M.Abramovici, "Built-in self-test of FPGA interconnect", *Proc. International Test Conference*, pp 404-411, 1998.
- [19] X.Sun, J.Xu, B.Chan, P. Trouborst, "Novel Technique for Built-In Self-Test of FPGA Interconnects", *Proc. IEEE International Test Conference*, pp. 795-803, 2000.
- [20] M.B.Tahoori, "Diagnosis of Open Defects in FPGA Interconnects", *Proc. IEEE International Conference on Field-Programmable Technology*, pp. 328-331, 2002.
- [21] M.B. Tahoori, S. Mitra, "Automatic Configuration Generation for FPGA Interconnect Testing," *Proc. of VLSI Test Symposium*, pp. 134-139, 2003.
- [22] Xilinx Corp. "Xilinx Datasheets", [www.xilinx.com](http://www.xilinx.com).