RECONFIGURATION AND FINE-GRAINED REDUNDANCY FOR FAULT TOLERANCE IN FPGAS

Nicola Campregher[§], *Peter Y. K. Cheung*[§], *George A. Constantinides*[§] *and Milan Vasilko*[‡]

[§]Department of EEE, Imperial College, London, UK
[‡]School of Design, Engineering and Computing, Bournemouth University, UK

ABSTRACT

As manufacturing technology enters the ultra-deep submicron era, wafer yields are destined to drop due to higher occurrence of physical defects on the die. This paper proposes a yield enhancement scheme based on the use of spare interconnect resources in each routing channel to tolerate functional faults. By using a node-covering technique and integer-linear programming (ILP) methods, the scheme is shown to provide minimal area and timing overheads. Significant yield improvements can thus be achieved.

1. INTRODUCTION

The area occupied by wiring channels and interconnect configuration circuits in an FPGA is significant, occupying 50 to 90 percent of the chip area [1]. With current trends aiming to reduce the area occupied by wiring segments in the routing channels, wire width and wire spacing have been reduced. This has in turn led to higher occurrences of wiring defects, such as breaks and shorts, and decrease in manufacturing yield [2] and fewer functioning devices at fixed manufacturing costs.

The nature of FPGA devices provides two separate ways of dealing with defects. The first, more obvious, method is based on hardware redundancy, and it capitalizes on the high regularity of the FPGA array to swap a faulty resource with a spare functioning one. The second method is based on exploiting the reconfiguration properties of the device, tweaking the design to fit around the defective resource. Both methods come at great area and timing overheads; only a limited number of the proposed schemes have proved successful and have been implemented by manufacturers [3, 4, 5].

In this work we propose a new fault tolerant scheme based on both reconfiguration and hardware redundancy. We propose a new approach to fault tolerance based on modifying some of the underlying characteristics of a given FPGA architecture, and we demonstrate the principle using a simple routing architecture.

Our fault tolerance scheme has the following notable advantages

This work is partially supported by EPSRC grant no. EP/C549481/1.

- Estimated device yields increase from 40% to 85% for large devices built at 90nm as predicted by our yield analysis tool [2].
- Estimated worst case timing degradation of 8.5%.
- Semi-permanent defect correction through configuration readback.
- Support for multiple non-localized defects.
- Can be extended to support dynamic fault tolerance.

The major disadvantage of this fault tolerance scheme comes in the need for a small configuration controller, which requires silicon area and will slow down the power-up sequence. At the time of publication an in-depth analysis of the overheads of the configuration controller has not been performed: these however are likely to be outweighed by the considerable yield advantages achieved by this scheme.

This paper is structured as follows. Section 2 provides a brief account of some of the most successful work carried out in the field of fault tolerance in FPGA. Section 3 introduces the fault tolerance scheme and gives details on the implementation. The results of our pilot architecture are presented in Section 4, and finally Section 5 concludes the paper and suggests area for future research in the area.

2. PREVIOUS WORK

Research carried out in the field of FPGA fault tolerance can be divided into two main categories. The first exploits the highly regular structure of the FPGA array. The second is based on the reconfiguration properties of FPGA devices.

In the first category, one of the earliest work was proposed in [6]. The research first analyzed the possibility of modifying the FPGA row selector and extending it to support the swapping of an entire row for a spare one in the case of a fault. Other methods, based on a finer redundancy, were proposed by two different authors in [7, 8]. Both these works propose widening the routing channels to include a spare interconnect to be swapped for a defective one. The main difference between these works is in the swapping procedure: the earlier work utilizes fuse blowing, while the second work uses a more sophisticated technique to shift all routing tracks in a channel using a more elaborate switch matrix.

Many approaches have been presented that exploit the reconfiguration properties of FPGA devices. The most comprehensive, based on roving areas to take part of the chip off-line, test and repair it has been presented in [9]. A method based on pre-compiled partial configurations has been proposed in [10], whereby segments of the design are placed and routed multiple times and then the configuration that avoids the defect is chosen for programming the device.

3. FAULT TOLERANCE SCHEME

This section describes the proposed fault tolerance scheme, its motivation and line of thought that brought to its development.

3.1. Motivation

The occurrence of defects during manufacturing is a random process. While data exist regarding the density and clustering of defects, it is impossible to formulate prediction models regarding the location of the defects within a die. As such, the probability of obtaining even two defective devices which exhibit the same functional fault in exactly the same location is almost non-existent.

As FPGAs continue their expansion into the semiconductor market, they are more and more often utilized in medium volume products. One of the biggest challenges offered by fault tolerance is therefore ensuring that the same bitstream produced can be successfully matched to tens, hundreds, and potentially even thousands of non-identical devices. Therefore the development of a fault tolerant scheme has the following primary goals:

- increase number of usable devices from wafers,
- ensure no extra design burden for the customer,
- maintain timing enclosure.

Area, despite being an important issue for most research, is not mentioned as a primary goal. This is because in order to increase the number of usable devices obtained from the manufacturing process the yield advantage has to overcome the area overhead. It is therefore more important to increase yields *and* reduce area overheads.

Under these constraints the most reasonable approach is to automatically manipulate the design before programming the FPGA. By generating *ad-hoc* bitstreams for each device the "uniqueness" factor of fault tolerance is eliminated. This can be achieved either through a configuration controller or through on-board placer and router, to be run with prior knowledge of the fault location [11]. However, other factors affect these type of approaches, most notably the overhead required to manipulate the bitstreams in a reasonable amount of time before programming. With designs getting more and more complex a full re-generation of bitstream is infeasible.

Therefore hardware redundancy, coming with greater area and timing penalties, has been the preferred method for implementing fault tolerance. As device performance is improved as a result of manufacturing technology and architectural improvements, the timing degradation resulting from the extra switching required to avoid the fault can be within acceptable limits for non-performance-dependent applications. Any form of hardware redundancy does, however, restrict the performance of devices, and in a fast moving semiconductor industry even the smallest degradations are crucial.

3.2. Preliminary analysis

The first step of our analysis was conducted to understand exactly how the highly redundant nature of FPGA affected the probability of a design being successfully placed and routed. VPR, an open source place and route tool [12], was modified in order to provide *fault injection*. This is achieved by making an interconnect resource unavailable to the router, to simulate the presence of a catastrophic interconnect fault. Selected benchmarks from the MCNC suite [13] have been placed and routed in a *minimal* FPGA (minimum array and channel width), then placed and routed in a *faulty* FPGA (FPGA without an interconnect resource). The tool was run once for every track used by the original design, so as to simulate operation under the presence of an interconnect fault. The results were split in 3 categories:

- *Successful* - The design was successfully placed and routed with the same timing characteristics of the original design.
- *Timing failure* - The design was successfully placed and routed but the timing was affected when compared to the original design.
- *Failed* - The design could not be successfully placed and routed.

The benchmarks were placed and routed using two routing architectures. The first, labelled *single* is a full connectivity, low performing architecture: this is the ideal fault-tolerant architecture because of the high routing flexibility. The second architecture taken into consideration is a segmented one, labelled *seg* which resembles commercial FPGA architectures, and is a compromise between routability and performance. Details of these two routing architectures will follow later.

Figure 1 shows the outcome of the place and route analysis. In the worst-case scenario of maximum array usage the percentage of faults causing the design to fail routing can be as high as 60% for the segmented architecture. The spread of timing variation is shown in Figure 2, and in both architectures some designs successfully place and route but exhibit very high timing degradation.

On average, the *single* architecture only fails the place and route process in under 10% of cases. However, if the

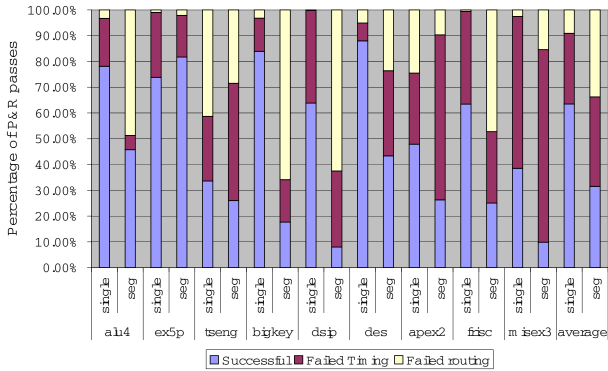

Fig. 1. Design classification in the presence of a fault

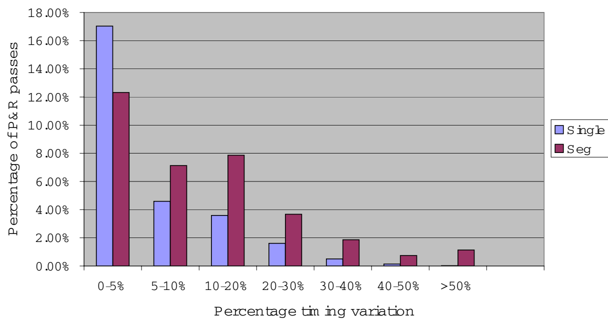

Fig. 2. Spread of timing variation for designs successfully placed and routed but with timing failure

experiment is repeated using one extra track in each routing channel, all designs route successfully and within timing constraints.

Figure 3 shows a portion of the *single* architecture. The striking feature of this type of architecture is the high connectivity from all pins of connection block (a) (A,B,C,D,N) to all pins of connection block (b) (E,F,G,H,O). More importantly, due to the nature of the architecture it is possible to replicate all pin-to-pin connections by simply widening the routing channel and introducing spare resources, as shown in Figure 3. It is this feature which enables designs to be placed and routed while maintaining similar timing charac-

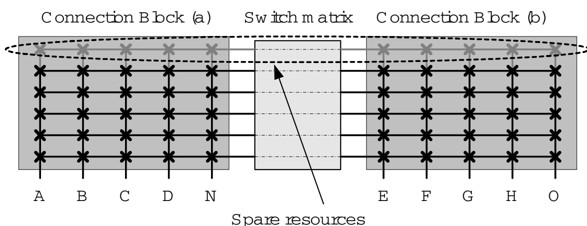

Fig. 3. *Single* architecture connectivity

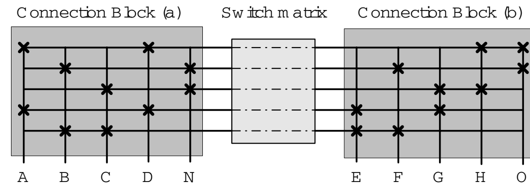

Fig. 4. An example architecture with limited connectivity

teristics to the original design. Duplicating pin-to-pin connections has thus been identified as being the key to achieving fault tolerance without disrupting the timing.

This brings the question of how much redundancy is really needed in FPGAs? Our analysis has shown that there is no need to introduce many more spare resources, rather it is convenient to maximize and improve the use of the available ones left over by the placer and router, and perhaps introduce spares at very fine grain level. This has brought the development of our fault tolerant architecture, discussed in the next subsection.

3.3. The proposed fault tolerant FPGA

The highly regular structure and high connectivity of the *single* architecture enables all pin-to-pin connections to be replicated with relative ease. Full connectivity, however, is subject to larger area and considerable timing penalties. High performance architectures thus have limited connectivity, as shown in the sample architecture in Figure 4; the architecture is designed taking into account the routability and performance only. We propose here to re-evaluate the connectivity of FPGA devices taking into account fault tolerance as a parameter.

Consider the architecture shown in Figure 4. The connectivity in each connection block and the switch matrix can be expressed mathematically using adjacency matrices, as shown in Figure 5. A matrix entry of “1” signifies a connection between a pin and an interconnect or an interconnect to another interconnect exists in a connection block or switch matrix respectively. Conversely, a matrix entry of “0” means no connection exists. The product of all three matrices yields the overall connectivity matrix, where each entry indicates the number of possible routes to connect each pin to all others.

In order to improve fault tolerance we aim to re-evaluate the overall connectivity matrix by increasing each non-zero matrix entry by one or more, and derive the individual adjacency matrices accordingly. Integer Linear Programming was used to solve this problem, and the formulation is shown in the next subsection.

3.3.1. ILP formulation

This section introduces the Integer Linear Programming (ILP) to split an overall connectivity matrix into the product of

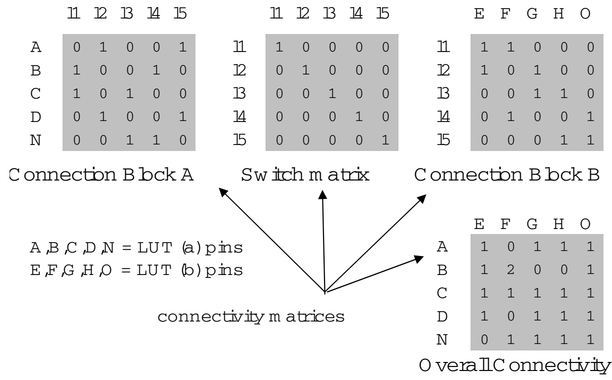

Fig. 5. Modelling connectivity using adjacency matrices

a three adjacency matrices, each representing a connection block or a switch matrix. In order to linearize the problem the formulation has been split into two stages, where in the first stage the overall connectivity matrix is split into the product of a binary permutation matrix and an intermediate matrix; in the second stage the intermediate matrix is further divided into two binary permutation matrices. The formulation of the first stage is shown in this section, it depicts the most general case of the problem.

Consider an architecture where each routing channel is t tracks wide and each LUT has p pins which connect to the routing channel. The overall connectivity matrix for this kind of architecture is modelled using a matrix C of p rows by p columns, which is the product of a binary permutation matrix A and an arbitrary matrix B of $p \times t$ and $t \times p$ rows and columns respectively. The main constraint is shown in (1).

$$\forall i, \forall j \sum_{k=1}^{t} a_{ik} b_{kj} \geq c_{ij} + 1. \tag{1}$$

As both a and b are variables in our system, we introduce a dummy variable d constrained by (2) to regulate (1) in linear form. Substituting for d yields (3).

$$d_{ikj} \leq a_{ik} b_{kj}. \tag{2}$$

$$\sum_{k=1}^{t} d_{ikj} \geq c_{ij} + 1. \tag{3}$$

Considering $a_{ik} \in \{0, 1\}$, (1 can be replaced by (4), which is then linearly expressed by (5) and (6), where U is an upper bound on b.

$$\begin{aligned} a_{ik} = 0 &\Rightarrow d_{ikj} \leq 0 \\ a_{ik} = 1 &\Rightarrow d_{ikj} \leq b_{kj}. \end{aligned} \tag{4}$$

$$d_{ikj} \leq a_{ik} U. \tag{5}$$

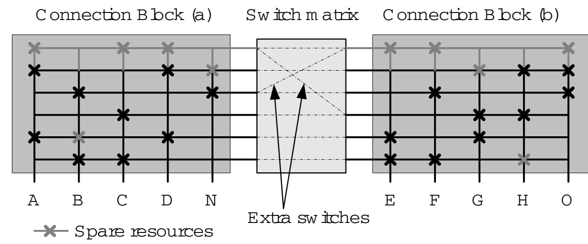

Fig. 6. Example architecture with improved connectivity for fault tolerance

$$d_{ikj} \leq b_{kj}. \tag{6}$$

Finally, in order to preserve the regular structure of the FPGA, the algorithm needs to ensure that all pins connect to a maximum number of interconnects and vice versa all interconnects only connect to a fixed number of pins. These conditions are expressed by (7) and (8), where N_1 and N_2 are the intended number of connections present, if a is a connection block adjacency matrix. Similar constraints are used for switch block permutation matrices.

$$\sum_{i} a_{ik} \leq N_1 \tag{7}$$

$$\sum_{k} a_{ik} \leq N_2 \tag{8}$$

The aim of formulation is to obtain an optimized solution to minimize area. The ILP objective is thus minimize the number of non-zero entries in the adjacency matrices, as they represent the number of switches present in each connection block and switch block. Considering the fact that transistors in connection blocks and switch matrices are likely to have different properties, the size and performance of each was based on the work presented in [14]. The final objective equation is shown in (9), where T_1 and T_2 are the relative sizes of the transistors in the resource being modelled by matrices A and B.

$$\min \sum_{i,k} (T_1 a_{ik}) + \sum_{k,j} T_2 b_{kj} \tag{9}$$

The final result of the ILP problem solving for the sample architecture shown in Figure 4 is shown in Figure 6. Due to limitations in space, only connections in the horizontal channels are shown. For a complete architecture the vertical connections are also considered.

3.3.2. Fault avoidance

The fault avoidance is based on a node covering scheme. Each point to point connection is "covered" by another option, so that if a track becomes unavailable as a result of

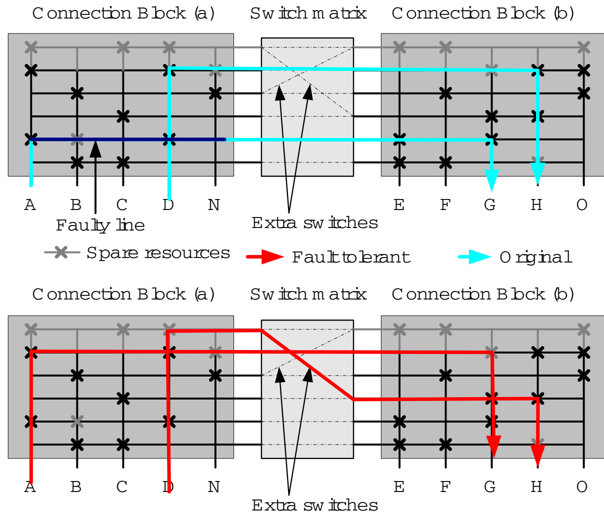

Fig. 7. Example of fault avoidance through re-routing

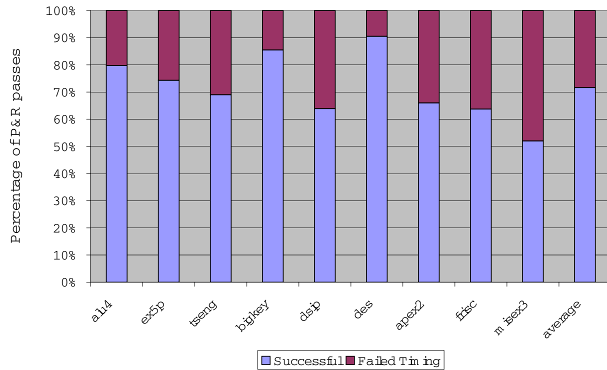

Fig. 8. Percentage of faults causing design degradation

fault, the "covering" track is used instead. The transformation required is very simple, and can easily be performed during bitstream loading. An in depth analysis of the bitstream controller is planned for future work.

An example transformation is shown in Figure 7. The original design, depicted on the top diagram, utilizes a faulty track (second from the bottom of connection block (b)). The transformation algorithm automatically swaps the signal to utilize a different track. This in turn "knocks" another signal, which originally utilized the covering track, onto its own cover. The final result is depicted on the bottom diagram of Figure 7.

4. ANALYSIS

Our fault tolerant architecture, developed using the technique presented in Section 3.3.1 applied to the *seg* architecture allows all of the designs taken into account to be placed and routed successfully, as shown in Figure 8. On average, over 70% of faults did not affect the designs in any way. The remaining faults only caused minor timing variations,

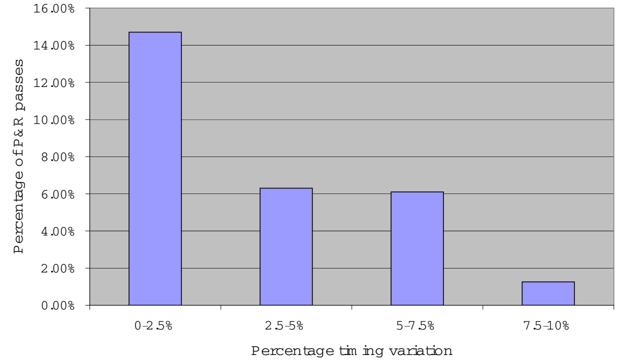

Fig. 9. Percentage of faults causing timing variations

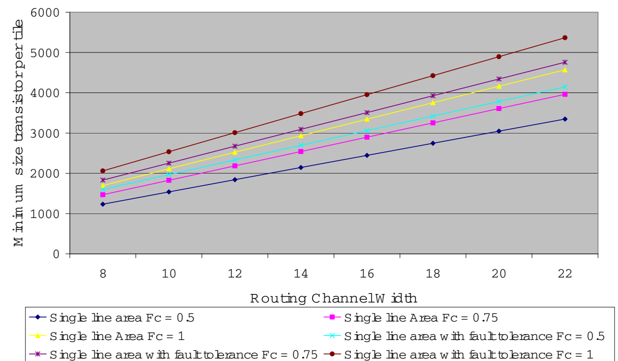

Fig. 10. Routing area analysis of fault tolerant architectures

as shown in Figure 9. All of the timing variations are within 8.5% of the original design, while the majority is contained within 5%.

Figure 10 depicts the area overhead incurred by the fault tolerant architecture. The curves shown depict the total routing area required to implement the architecture with and without fault tolerance. The graph also shows how varying the parameter F_C, the fraction of pins each track connects to, affects the overall area requirements ($F_C = 1$ means the the interconnect track connects to all pins, while $F_C = 0$ means the track does not connect to any pins). The total area is shown in minimum transistor size count, an area measuring technique introduced in [14]. This method takes into account the different sizes of connection block, buffers, and switch blocks transistors, and it offers a much more realistic measure of the total area required to implement an architecture. For the analysis, no buffer sharing was assumed, *i.e.* every connection needs an individual buffer.

The results shown in Figure 10 only depict the routing area requirements. In order to calculate the total tile area the transistor counts of the logic block would need to be included. Our sample routing architecture is, for a routing channel width of 16, 31% larger than the *seg* architecture which it generates from. When logic is included, this value

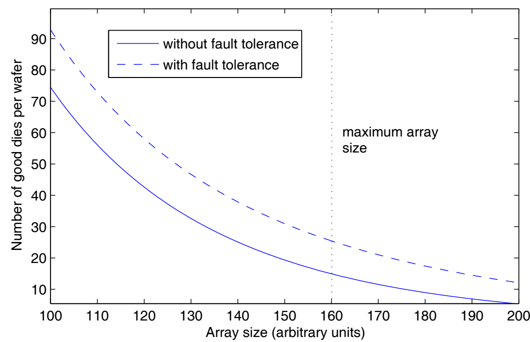

Fig. 11. Using the proposed fault tolerance scheme can almost double the productivity for very large devices at 90nm

is reduced to 19%.

The rather large area overhead is however overshadowed by the great yield increase, and reflected in the total number of working devices out of a wafer. Figure 11 shows the variation in number of working dies per wafer as a function of array size, shown here in arbitrary units. The largest devices built at 90nm, the technology being analyzed here, is shown by the vertical dotted line in Figure 11. If no hardware redundancy is used, 14 working dies can be expected using this fault tolerance scheme. Using our fault tolerance scheme the total number of working dies can be increased to 26, thereby almost doubling the productivity. Details on the yield analysis framework used here can be found in [2].

5. CONCLUSIONS

A new approach to fault tolerance in FPGAs has been presented. The method proposes to re-evaluate the routing architecture of FPGA devices to include fault tolerance as a measuring parameter as well as performance and routability.

The scheme proposed is based on node-covering techniques to replace faulty tracks by spare ones. Area overhead is limited by minimizing the number of extra switches required to implement the node-covering.

It has been shown that even in the worst-case scenario timing variation is within 8.5% of original design. Using yield analysis techniques presented in past research it has also been possible to prove that yields can be increased significantly, almost doubling the total number of working dies per wafer despite the area overhead.

The scheme requires a configuration controller to implement the fault avoidance. At time of publication an extensive study on the performance of the controller has not be carried out. Future work will also include exploring details of efficient Built-in-Self-Test techniques to identify cheaply and quickly the location of faults in the FPGA.

6. REFERENCES

[1] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field Programmable Gate Arrays*, MA: Kluwer, 1992.

[2] N. Campregher, P.Y.K. Cheung, G.A. Constantinides and M. Vasilko, "Analysis of yield loss due to random photolithographic defects in the interconnect structure of fpgas," in *Thirteenth ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2005.

[3] "Altera Says Redundancy Technology Increases Yields," 2000. [Online]. Available: http://www.reed-electronics.com/electronicnews/article/CA94271.html

[4] C. McClintock, A. L. Lee, and R. G. Cliff, "Redundancy circuitry for logic circuits," 2000.

[5] S. T. Reddy, M. Mejia, A. L. Lee, and B. B. Pedersen, "Programmable logic device with redundant circuitry," 2002.

[6] F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki, "Introducing redundancy in field programmable gate arrays," in *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, 1993, pp. 7.1.1–7.1.4.

[7] F. Hanchek and S. Dutt, "Methodologies for tolerating cell and interconnect faults in FPGAs," *IEEE Transactions on Computers C*, vol. 47, no. 1, pp. 15–33, 1998.

[8] A.J. Yu and G.G.F. Lemieux, "Defect tolerant fpga switch block and connection block with fine-grain redundancy for yield enhancement," in *Proceedings of FPL 2005. 15th International Conference on Field Programmable Logic and Applications*, Tampere, Finland, 2005, pp. 255–262.

[9] M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving STARs: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems," in *NASA/DoD workshop on evolvable hardware*, D. Keymeulen, Ed. Long Beach, CA: IEEE Computer Society, 2001, pp. 73–92.

[10] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," in *Field programmable gate arrays; FPGA '00ACM/SIGDA*. Monterey, CA: Acm, 2000, pp. 187–194.

[11] S. Dutt, V. Shanmugavel, and S. Trimberger, "Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays," in *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers, 7-11 Nov. 1999*, ser. 1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051). San Jose, CA, USA: IEEE, 1999, pp. 173–6.

[12] V. Betz and J. Rose, "Vpr: a new packing, placement and routing tool for fpga research," *Field-programmable Logic and Applications. 7th International Workshop, FPL '97. Proceedings*, pp. 213 – 22, 1997.

[13] S. Yang, "Logic synthesis and optimization benchmarks, version 3.0," *Microelectronics Centre of North Carolina*, 1991.

[14] V. Betz, "Architecture and cad for speed and area optimization of fpgas," PhD Thesis, University of Toronto, 1998.