

DEFECT TOLERANCE IN MULTIPLE-FPGA SYSTEMS

Zohair Hyder, John Wawrzyniek

Electrical Engineering & Computer Science
University of California - Berkeley
{zhyder,johnw}@eecs.berkeley.edu

ABSTRACT

SRAM-based FPGAs have an inherent capacity for defect tolerance. We propose a simple scheme that exploits this potential in multiple-FPGA systems. The symmetry of the system is exploited to yield a large number of possible mappings of bitstreams on FPGAs, which results in a high probability that at least one functional mapping exists. We show that the behavior of a system built using a large number of defective FPGAs approaches that of the ideal defect-free system. Various interconnection topologies such as the tree, the crossbar, and a hybrid form are compared.

1. INTRODUCTION

Various defects may be produced in a VLSI chip during manufacturing. The existence of defects affects yield and ultimately cost. FPGAs are especially expensive due to the large area penalty taken in favor of reconfigurability. Unlike ASICs however, FPGAs have an inherent potential for defect tolerance. Since any user design uses only a fraction of the FPGAs resources, the reconfigurability of FPGAs can be exploited to bypass defective parts of the chip. In the literature, the terms “defect” and “fault” are often used interchangeably; the former refers specifically to permanent faults such as those produced during manufacturing.

Several defect tolerant approaches have been suggested, any of which might work on a single FPGA system. We propose a simple scheme that can effectively be applied to multiple-FPGA systems. A multiple-FPGA system consists of several communicating FPGAs. Each of these FPGAs, when configured with the appropriate bitstream, contributes towards the common goal of the system. In general, the system may be reconfigured for different applications.

If the interconnection topology of the system possesses symmetry, there may be multiple possible ways to map the bitstreams onto the FPGAs. For example, if two FPGAs are connected to the rest of the system in a fashion such that they are indistinguishable, then their bitstreams can be swapped. For the system to work as desired, only one mapping is required to be functional. A functional mapping of bitstreams on FPGAs is a mapping in which no bitstream

utilizes the defective resources of the FPGA it is mapped to. The larger the number of ways to perform mapping, the greater the probability that at least one functional mapping exists.

Section 2 describes related work. Section 3 shows how defects can be detected. Section 4 illustrates the basic concept of bitstream swapping. Section 5 describes different interconnection topologies and their associated implementation overheads. Section 6 shows the probability of achieving a successful mapping in each of the topologies, and Section 7 describes the relationship between yield and these probabilities. Finally, Section 8 concludes the paper.

2. RELATED WORK

Several approaches have been suggested for the attainment of fault and defect tolerance. Solutions range from architectural additions to CAD tool modifications. [10] and [9] suggest row/column swapping and node covering respectively; the latter promises much lower area overhead than the former by operating at a finer granularity. Both these approaches require architectural changes to accommodate spare resources along with significant flexibility in routing resources. [7] present a scheme that achieves fault tolerance by horizontally and vertically shifting the configuration data. [1] suggest BIST-based techniques for tolerating defects, which involves “roving STARS”, self-test areas, that move around the chip by using partial runtime reconfiguration. A defect detected within a STAR results in the entire STAR being flagged as unusable by the user’s design. This approach, however, requires several architectural features to enable BIST as well as to allow swapping of partial configurations. [8] require a combination of CAD tool and architecture changes to enable fast incremental routing around defects. The HP Teramac [4] is a defect tolerant multiple-FPGA system, in which the techniques used for achieving defect tolerance are primarily CAD tool-based. Defect maps are maintained for each FPGA and the place-and-route tool generates bitstreams that bypass defects.

Our approach is unique in that it can be implemented using off-the-shelf FPGAs and CAD tools with no modifi-

cation. A simple additional tool is required for testing, as described in Section 3. System integration (board design, etc.) may require slightly greater effort than usual. Still the system is easily viable; our research group is currently building a defect-tolerant reconfigurable supercomputer using Xilinx Virtex-II Pro FPGAs and the standard Xilinx CAD tool flow.

3. DEFECT DETECTION

To enable our defect tolerant approach, we need to detect if a given bitstream uses any defective resources in a particular FPGA. If we have n bitstreams and n FPGAs, in general we need to perform defect detection for each of the n^2 bitstream-FPGA pairs in order to find a functional mapping. Note that the n FPGAs may run tests in parallel, so applying tests for the n^2 pairs can be done in $O(n)$ time. Testing for defect detection can be accomplished through either of two methods: full chip testing or bitstream-dependent testing.

3.1. Full Chip Testing

Full chip testing involves comprehensive testing of the entire FPGA at manufacturing time so as to generate a *defect map*. The defect map indicates for each resource on the FPGA whether it is defective or defect-free. Note that this testing requires accurate defect localization. Each of the n FPGAs in a system will have unique defect maps. When the system is to be configured for a certain application, we must perform comparisons between n bitstreams and n defect maps in order to accomplish defect detection. In the context of this section, “defect detection” does not refer to the defects detected during full chip testing; rather it refers to the defects detected within a bitstream during defect map comparisons.

Several approaches for full chip testing have been suggested in the literature. [11, 12] propose techniques for testing CLBs using ILA techniques. While testing CLBs is straightforward and well understood, testing interconnect has traditionally been more difficult. [13] offers a C-testable approach to testing interconnect. [1] suggests a BIST-based approach that promises to test all of an FPGA’s resources.

3.2. Bitstream-Dependent Testing

Full chip testing can be a daunting task, especially in the context of our system which uses off-the-shelf FPGAs. Neither do we have knowledge of the FPGA’s layout, nor do we have the ability to use design-for-testability (DFT) features of the chip. Without knowledge of the layout, we cannot create a fault list for bridging faults. In the absence of DFT, we would have to test the FPGA with an unmanageable number of bitstreams, since each bitstream would only use a fraction of the resources of the chip. Interconnect resources in particular are difficult to test, since they occupy more than

90% of a contemporary FPGAs’ area, but any bitstream may utilize only 5-10% of these resources.

Bitstream-dependent testing involves testing only the specific resources used by the bitstream at configuration time, rather than at manufacturing time. Since only a single placed-and-routed bitstream needs to be tested, there is no issue of explosion in the number of bitstreams. Further, bitstream-dependent testing does not require defect localization. Xilinx’s EasyPath flow involves bitstream-dependent testing, but TPG and testing take several weeks since the target market is that of non-reconfigurable ASIC replacements. Our goal is to incur a small TPG and testing overhead that is negligible compared to place-and-route time, enabling a truly general purpose reconfigurable system. [6, 15, 14] present approaches for bitstream-dependent testing of interconnect that require minimal TPG runtime. In [14], all stuck-at faults and all of $m \cdot (m-1)/2$ possible bridge faults are tested using only $\log_2 m$ number of configurations, where m is the number of nets in the bitstream. The basic idea of the approach involves exploiting the configurability of look-up tables and other FPGA resources to achieve near-ideal controllability and observability in the circuit. The interconnect configuration itself is left untouched. The details of this test approach are outside the scope of this paper.

We are also working on a bitstream-dependent testing approach for delay faults. Preliminary results in fault simulation are encouraging. Bitstream-dependent TPG can be completed in time of the order of 10 seconds for circuits with up to 4000 LUTs, though TPG times would vary with respect to size of the circuit. TPG complexity is no greater than $O(m \cdot \log_2 m)$. Our complete test strategy involves testing CLBs using techniques described in Section 3.1 and testing interconnect using bitstream-dependent methods.

4. BITSTREAM SWAPPING

Bitstream swapping is the central idea behind our defect-tolerant strategy. The ability to swap bitstreams results in multiple ways to perform mapping, only one of which needs to be functional. To enable bitstream swapping, it is necessary that the system possess symmetry. Two or more FPGAs must appear identical to the rest of the system if bitstreams are to be swapped between them.

An FPGA may have multiple ports for connecting to other FPGAs. Each port has an identical number of IO pads with identical IO capabilities. Each bitstream also includes a block containing several muxes that allows swapping between the multiple ports. Further the SRAM cells, either in LUTs or in BRAM, that define the mapping of ports are locked at specific resources within the FPGA, so that the values in these cells can be changed without having to redo synthesis or place-and-route. We’ve implemented a tool that can directly modify the bitstream of Xilinx Virtex-II Pro FP-

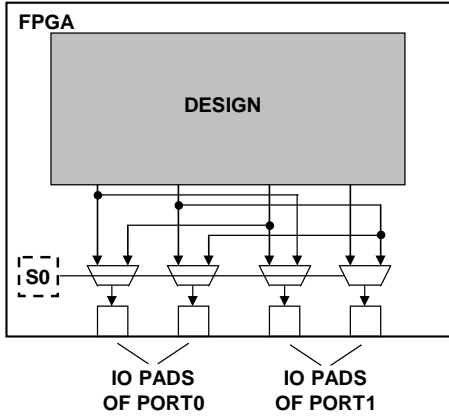


Fig. 1. A two-port FPGA.

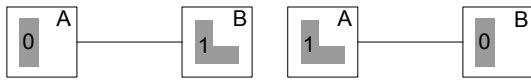


Fig. 2. Swapping with two FPGAs.

GAs for this. Figure 1 describes an FPGA that has two ports, in which the ports can be swapped by writing into SRAM cell S0.

Figure 2 illustrates the basic concept for a two-FPGA system. Clearly two mappings are possible. In the first, bitstream 0 maps to FPGA A, and 1 to B; in the second, 0 maps to B, and 1 to A.

Figure 3 shows an example of a three-FPGA angle system. Note that in this case, only two mappings are possible since just the leaf nodes may swap bitstreams with each other, as bitstream 0 needs to be connected to both 1 as well as 2, while 1 is not directly connected to 2. However, the triangle system shown in Figure 4 allows six different mappings. Given the more favorable defect tolerant behavior of the triangle system, one could argue that this topology should be selected over the angle topology, even if it is known a priori that 1 and 2 will not need to communicate with each other. In general, we argue in favor of topologies that possess greater symmetry than the minimum needed for communication in the application.

Let p be the base probability that any given bitstream will work on any FPGA. We assume that this probability is independent across FPGAs, i.e. defects occur at different locations in different FPGAs. We feel this condition can be

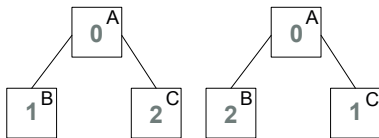


Fig. 3. Swapping with three FPGAs connected as an angle.

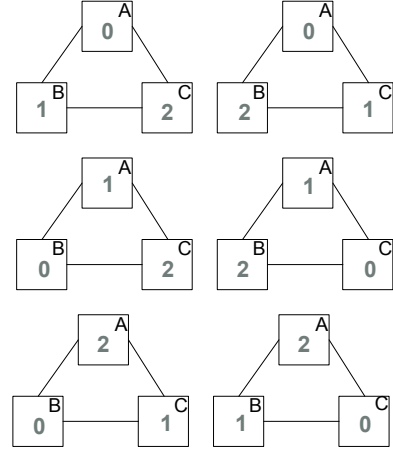


Fig. 4. Swapping with three FPGAs connected as a triangle.

Table 1. Success Probabilities For 3 Simple Systems.

| System | p | P w/o Swapping | P w/ Swapping |
|--------------------|------|------------------|-----------------|
| 2-FPGA | 0.25 | 0.063 | 0.121 |
| | 0.5 | 0.25 | 0.438 |
| | 0.75 | 0.563 | 0.809 |
| 3-FPGA Angle | 0.25 | 0.016 | 0.03 |
| | 0.5 | 0.125 | 0.219 |
| | 0.75 | 0.422 | 0.606 |
| 3-FPGA Triangle | 0.25 | 0.016 | 0.084 |
| | 0.5 | 0.125 | 0.482 |
| | 0.75 | 0.422 | 0.904 |

sufficiently satisfied by selecting FPGAs from different lots, etc. Further we also assume that p is independent across bitstreams, i.e. different bitstreams use different resources of an FPGA. This condition too can be easily satisfied by restricting the nature of applications, e.g. applications using cellular automata must be disallowed. Our aim is to maximize the success probability P of at least one mapping being functional. In most cases, a larger number of possible mappings implies a greater P . However in general there isn't an absolute correlation between number of mappings and success probability, since the probability of any mapping being functional need not be independent of the probabilities of other mappings being functional.

Table 1 shows the success probability of the three systems mentioned above, with and without swapping. Note how swapping can make a dramatic difference in the success probability. The triangle system has a substantially better success probability than the angle system. Further, P is greater for the triangle system than the two-FPGA system for $p = 0.75$ suggesting that scaling up such highly symmetric systems improves the success probability.

5. INTERCONNECTION TOPOLOGY

The interconnection topology can have a significant impact on the defect tolerant behavior of the system. Highly symmetric topologies can have the success probability approach unity with increase in the size of the system. On the other hand, topologies with poor symmetry can yield nearly zero success probability for large systems.

In this section, we will analyze the behavior of topologies with good symmetry properties, namely the tree and the crossbar. Several other topologies in common use are not analyzed in this paper. Examples of topologies with poor symmetry include the token ring, and meshes and toruses with low dimensionality. Topologies with greater symmetry include meshes or toruses with high dimensionality, and boolean n-cubes.

5.1. Crossbar

The crossbar topology possesses maximal symmetry. Every node is connected to every other node in an identical fashion. Hence, any node may be swapped with any other node. There are $n!$ mappings possible for an n -FPGA system. However, this symmetry comes at a quadratic cost in implementation. For smaller systems, the overhead may still be reasonable. For larger systems, a hybrid topology as described in Section 5.3 may be more appropriate.

We assume that the interconnection resources used to implement the crossbar are defect-free. In actual systems, these would include the cables, connectors, PC cards, and switch chips. We assume that the crossbar is implemented using some switch, the routing tables of which can be quickly changed to reflect swapping of bitstreams.

The problem of mapping n bitstreams on n FPGAs is identical to that of perfect matching in bipartite graphs. While there are several ways of determining the perfect or maximal matching in bipartite graphs, the use of the max flow Ford-Fulkerson method yields runtime of $O(|V| \cdot |E|)$, whereas the use of the Hopcroft-Karp method results in runtime of $O(\sqrt{|V|} \cdot |E|)$ [3, 2].

5.2. Tree

A tree topology of FPGAs resembles the structure of a directed acyclic graph in which every node except the root has exactly one incoming edge, and no node has more than a outgoing edges where a is the arity of the tree. Such a tree is often referred to as an “ a -ary tree.” In the FPGA implementation, every edge is implicitly bidirectional.

For compact trees, tree depth $d = \lceil \log_a n \rceil$ where n is the number of leaf nodes. We assume that only the leaf nodes participate in computation, and are implemented using defective FPGAs. All other nodes serve to only enable communication between the computation nodes. Furthermore,

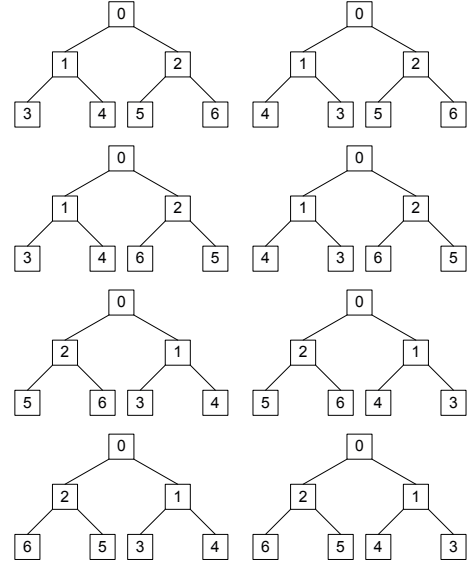


Fig. 5. Mappings in a binary tree.

for this study, we also assume that communication nodes are perfect, i.e. if communication nodes are implemented using FPGAs, these FPGAs must be defect-free.

Symmetry exists at every child-bearing node in the tree. For each parent node, its various children -upto a - are indistinguishable. Clearly increasing a would increase the symmetry of the system. At the limit however, when $a = n$ and $d = 1$, the tree reduces to the crossbar. The usual purpose of having a tree structure with $a < n$ is to reduce the overhead of implementing the system. As mentioned previously, crossbar systems have a quadratic interconnection overhead.

Figure 5 indicates how symmetry can be exploited to get a large number of possible mappings. If the communication nodes are implemented using FPGAs, these FPGAs also contain ports that are swappable by direct modification of their bitstreams. Hence swapping never requires synthesis or place-and-route to be repeated for communication FPGAs. Equation 1 gives the number of permutations about any parent node, assuming a children for every parent. Each parent node exhibits symmetry independent of other parent nodes in the system. Equation 2 gives the expression for the number of communication nodes needed. Hence the total number of mappings possible is given by the product of the numbers at each parent node, resulting in Equation 3.

If the application is communication latency insensitive, then there are no constraints regarding the locations for bitstreams within the tree. Hence all $n!$ permutations can serve as valid mappings resulting in the same symmetry as the crossbar. The general case, however, involves the constrained location of different bitstreams, e.g. in Figure 5 bitstreams 3 and 4 must be no more than one hop apart.

Table 2. Overhead in tree topology.

| Arity | Overhead (units/FPGA) | | |
|-------|-----------------------|---------|--------------|
| | $d = 1$ | $d = 2$ | $d = \infty$ |
| 2 | 2.0 | 3.0 | 4.0 |
| 3 | 3.0 | 4.0 | 4.5 |
| 4 | 4.0 | 5.0 | 5.3 |
| 5 | 5.0 | 6.0 | 6.3 |

$$n_{mappings_per_parent} = a! \quad (1)$$

$$n_{parent} = \lceil \frac{n-1}{a-1} \rceil \quad (2)$$

$$n_{mappings} = (n_{mappings_per_parent})^{n_{parent}} \quad (3)$$

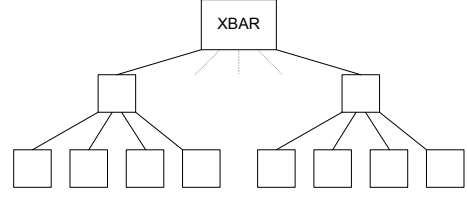
For each parent node in the system, the overhead of implementing the node is assumed to be quadratic to the number of edges connected to that node. In effect, each communication node acts like a $(a+1)$ -way crossbar. The overhead OH of the system is hence given by Equation 4. For all trees of any given arity, the overhead is linear to the size of the system. Hence this analysis would indicate that the tree topology is less expensive to implement than the crossbar, especially for large systems. Table 2 describes this overhead for a ranging from 2 to 5. In this table, each unit refers to one of x^2 units in an x -way crossbar; e.g. a 2-way crossbar would imply 4 units.

$$OH_{tree} = n_{parent} \cdot (a+1)^2 \quad (4)$$

Mapping in trees of a given arity, with constrained placement of bitstreams, is an $O(n^2)$ problem. Let the initial syndrome be the $n \times n$ boolean matrix indicating the success or failure of n bitstreams on n FPGAs. For each of the $\log_a n$ levels, a new syndrome needs to be developed, resulting in syndrome matrices of sizes $\frac{n}{a} \times \frac{n}{a}$, $\frac{n}{a^2} \times \frac{n}{a^2}$, and so on. For each of the elements in a non-initial syndrome, determining its value involves perfect bipartite matching at the lower level of the tree. Each instance of bipartite matching involves mapping a bitstreams on a FPGAs. Given the sizes of the multiple syndrome matrices, the number of bipartite matching instances is $= n^2/a^2 + n^2/a^4 \dots + 1 = \frac{n^2}{a^2} \cdot \frac{1-1/n^2}{1-1/a^2} = O(n^2)$.

5.3. BEE2 Reconfigurable Supercomputer

Our research group is currently building a reconfigurable supercomputer called the Berkeley Emulation Engine II (BEE2). The basic structure of the system involves a set of *modules*, each of which are implemented as a tree of fixed d and a .

**Fig. 6.** BEE2 system topology.

The root nodes of the modules are then interconnected using a full crossbar.

With this type of topology, the symmetry of the crossbar may be exploited, while simultaneously reducing the number of ways of the crossbar so as to achieve feasibility in implementation of even large systems. Effectively symmetry is traded off for cost and ease of implementation. Equation 5 describes the overhead of this topology.

$$OH_{BEE2_xbar} = \lceil n/a^d \rceil^2$$

$$OH_{BEE2_tree} = \frac{n - \lceil n/a^d \rceil}{a-1} \cdot (a+1)^2$$

$$OH_{BEE2} = OH_{BEE2_xbar} + OH_{BEE2_tree} \quad (5)$$

For our system, shown in Figure 6, we chose $a = 4$, and $d = 1$. A value of 4 was selected for a since it appeared to be the largest number of edges that could be supported by a single FPGA. This system results in an $n/4$ -way crossbar, which can be easily implemented using off-the-shelf InfiniBand switches even for $n > 256$. The overhead of the BEE2 is effectively about 1/16 that of a full crossbar.

6. SUCCESS PROBABILITY

In this section, we describe the defect-tolerant performance of the different interconnection topologies. Our aim is to maximize the success probability P . A value of P close to unity implies that almost every application will be functional, and the defect-tolerant system will appear virtually the same as one built out of defect-free FPGAs. All topologies described above show a threshold behavior for P w.r.t. base probability p .

For the crossbar topology, the problem of mapping n bitstreams onto n FPGAs is identical to that of perfect matching in bipartite graphs. The threshold number of edges that yields a perfect matching in bipartite graphs is known to be $n \cdot \log_2 n$ [3, 2]. The expected number of edges is $p \cdot n^2$, resulting in threshold probability $p_{th} = \log_2 n/n$. A very high P is obtained if p is greater than p_{th} , whereas a very low P is obtained if p is less than p_{th} . Figure 7 shows this threshold behavior for the simple crossbar as well as the BEE2 hybrid topology. For large systems, even a small p is sufficient to result in a near-unity P . In other words, the larger the system, the more defective the FPGAs may be.

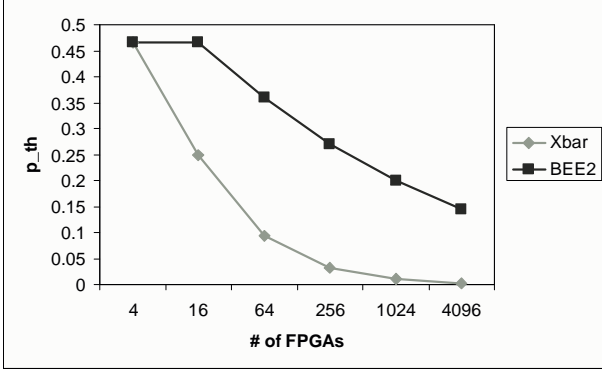


Fig. 7. Threshold probability for crossbar and BEE2 topologies.

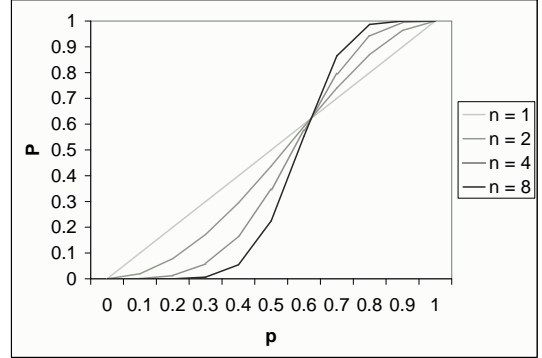
The BEE2 system has higher threshold probability than a simple crossbar for any given system size. However, the threshold probability trends are still encouraging. The primary concern in designing such systems should be that p_{th} is sufficiently low to allow typical FPGAs to be used. A system that offers a p_{th} of 0.01 may be overkill if manufacturing never produces FPGAs with p values that low. A higher defect density implies a lower value for p . In Section 7, we indicate what values of p one can expect.

Success probability follows a threshold function in the tree topology as well. However, the threshold probability does not decrease with increase in size of the system. Figure 8 shows P as a function of p for different system sizes, and trees of different arity. These probability results were obtained through brute-force counting. With increase in system size, the threshold function becomes sharper. The threshold probability is lower for the 4-ary tree than it is for the 3-ary tree, which is lower than it is for the binary tree. We would expect this trend since an increase in the arity of the tree produces greater symmetry.

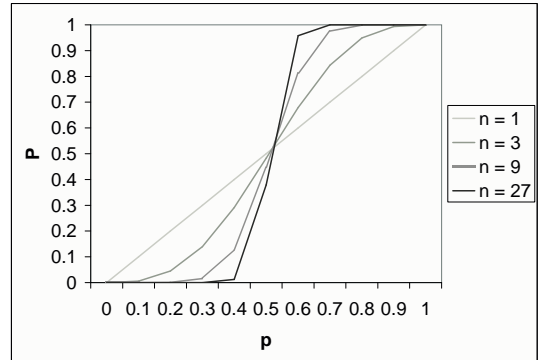
While we have yet to precisely quantify this relationship, it seems clear that there is an inherent tradeoff between the success probability and the implementation cost of the system. High symmetry topologies are expensive but lead to high success probabilities, and conversely low symmetry topologies, while inexpensive to implement, lead to few opportunities to avoid defects.

7. YIELD ANALYSIS

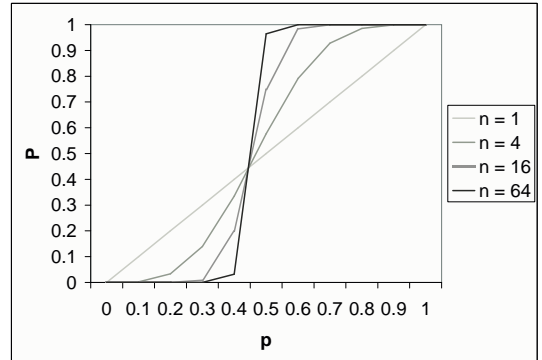
In this section, we analyze the effect of yields on the performance of systems built using our approach. While several yield models exist, we consider the classical Poisson model for yields [5]. Equation 6 shows the yield formula as derived from the Poisson model, where D represents the defect density per unit area, and A represents the area of the FPGA. We present a modified version in Equation 7, where



(a)



(b)



(c)

Fig. 8. Success probabilities for tree topologies: (a) binary; (b) 3-ary; (c) 4-ary.

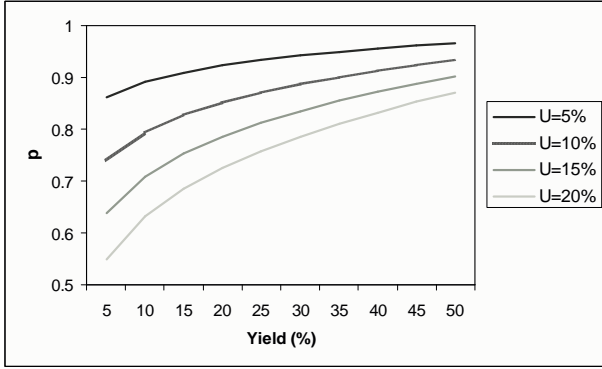


Fig. 9. Base probability as a function of yield.

Y_C represents the reduction in yield due to defects in critical resources, and A_{NC} represents the non-critical area of the FPGA. Critical defects may include power-to-ground shorts, bad IO pads, defective configuration circuitry, etc. FPGAs having such critical defects are unusable and are rejected. Y_{NC} represents the reduction in yield due to defects in non-critical resources such as CLBs, PIPs, etc. Defects in non-critical resources can be overcome by our defect-tolerant approach, effectively enhancing yield.

$$Y = e^{-D \cdot A} \quad (6)$$

$$Y_{NC} = e^{-D \cdot A_{NC}}$$

$$Y = Y_C \cdot Y_{NC} \quad (7)$$

Any bitstream utilizes only a small fraction of the FPGA's resources. Even if 70-80% of LUTs are utilized, only 5-10% of programmable interconnect points (PIPs) may be utilized. Since a majority of an FPGA's area is dedicated to interconnect resources to allow routability, the utilization fraction U is quite small. Equation 8 gives the relation for p , which is identical to that of Y_{NC} , except that A_{NC} is replaced by $A_{NC} \cdot U$.

$$p = e^{-D \cdot (A_{NC} \cdot U)}$$

$$p = Y_{NC}^U \quad (8)$$

Figure 9 describes the dependence of p on Y_{NC} for different utilizations. Note how even yields as low as 5% can result in high values of p . The values of p are sufficiently larger than the threshold probabilities needed in nearly all topologies described. In particular, the resulting value of p is satisfactory for the BEE2 supercomputer.

Since all topologies described exhibit strong threshold behavior for success probability, all FPGAs containing only non-critical defects (no critical defects) are rendered usable, and appear the same as defect-free FPGAs, i.e. $Y = Y_C$.

Hence, yield improves by a factor of $1/Y_{NC}$ using our defect tolerant approach. If the cost per chip is inversely proportional to yield, and the cost of the system is dominated by the cost of the FPGAs, overall cost also improves by a factor of $1/Y_{NC}$. In reality, cost of other components like PC boards, switches, etc. dominates beyond a certain point. To maximize cost savings, we must use FPGAs with the lowest yield that results in a sufficiently high p for the given system topology. If p is not sufficiently high, we may reduce U by restricting users to designing smaller circuits.

If needed, we can increase p by using only those FPGAs that have few defects, and rejecting those with a high defect density. Note that full chip testing is unnecessary; by simply testing the FPGAs with a set of random bitstreams, we can correlate the number of discovered defects with the number of actual defects. Of course, use of this strategy would result in lower cost savings.

8. CONCLUSION

We have presented a simple scheme that facilitates defect tolerance in multiple-FPGA systems. Using defective FPGAs can substantially drive down the cost of the system. Our analysis of success probability shows that large systems with high degrees of symmetry can appear nearly the same as defect-free systems. The only tradeoff may be the cost of extra communication resources in such topologies. Nevertheless with careful design of the system, the right balance between symmetry and implementation cost can be determined, and significant cost savings can be realized. We feel that defect tolerance must be considered as one of the key factors in determining the topology of a multiple-FPGA system, in addition to the communication requirements of the application. Even for applications that fit on a single FPGA, if performance is not critical, there may be a cost benefit to implementing the application on multiple FPGAs.

9. ACKNOWLEDGEMENTS

We would like to thank Prof. Christos Papadimitriou, members of the BEE2 research group, and Xilinx for their inputs, and help in conceiving and developing the ideas presented in this paper. This research was supported in part by the grant titled "Collaborative Research in the Design, Verification, and Test of Integrated Gigascale Systems: The Gigascale Systems Research Center," MARCO Contract #2003-DT-660.

10. REFERENCES

- [1] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriga, and V. Verma. Using roving star's for on-line testing and diagnosis for fpgas in fault tolerant applications. *Proc. International Test Conf.*, 1999.

- [2] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [4] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider. Defect tolerance on the teramac custom computer. *Proc. FPGAs for Custom Computing Machines*, 1997.
- [5] J. A. Cunningham. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Trans. on Semiconductor Manufacturing*, May 1990.
- [6] D. Das and N. A. Toubia. A low cost approach for detecting, locating, and avoiding interconnect faults in fpga-based reconfigurable systems. *Proc. of IEEE International Conf. on VLSI Design*, 1999.
- [7] A. Doumar, S. Kaneko, and H. Ito. Defect and fault tolerance fpgas by shifting the configuration data. *Proc. Defect and Fault Tolerance*, 1999.
- [8] S. Dutt, V. Shanmugavel, and S. Trimberger. Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays. *Proc. International Conf. on Computer Aided Design*, 1999.
- [9] F. Hanchek and S. Dutt. Methodologies for tolerating cell and interconnect faults in fpgas. *IEEE Trans. on Computers*, January 1998.
- [10] F. Hatori. Introducing redundancy in field programmable gate arrays. *Proc. IEEE Custom Integrated Circuits Conf.*, 1993.
- [11] W. K. Huang, M. Y. Zhang, F. J. Meyer, and F. Lombardi. A xor-tree based technique for constant testability of configurable fpgas. *Proc. Asian Test Symp.*, 1997.
- [12] T. Inoue, H. Fujiwara, H. Michinishi, T. Yokohira, and T. Okamoto. Universal test complexity for field-programmable gate arrays. *Proc. Asian Test Symp.*, 1995.
- [13] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. Testing the interconnect of ram-based fpgas. *IEEE Design and Test of Computers*, 1998.
- [14] M. B. Tahoori. Application-dependent testing of fpga interconnect. *Proc. Defect and Fault Tolerance*, 2003.
- [15] M. B. Tahoori. Using satisfiability in application-dependent testing of fpga interconnects. *Proc. Design and Automation Conference*, 2003.