

# Defect tolerance in multiple-FPGA systems

Z. Hyder and J. Wawrzynek

**Abstract:** SRAM-based field programmable gate arrays (FPGAs) have an inherent capacity for defect tolerance. A simple scheme that exploits this potential in multiple-FPGA systems is proposed. 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. It is shown that the behaviour 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. Field programmable gate arrays (FPGAs) are especially expensive because of the large area penalty taken in favour of reconfigurability. Unlike application specific integrated circuits (ASICs), however, FPGAs have an inherent potential for defect tolerance. As 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 for single FPGA systems. 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 utilises 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.

## 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. Hatori [1] and Hanchek and Dutt [2] suggests row/column swapping and node covering, respectively, the latter promises much lower area overhead than the former by operating at a finer granularity. Both of these approaches require architectural changes to accommodate spare resources along with significant flexibility in routing resources. Altera applies row/column swapping at manufacturing time to its FPGAs [3]. Doumar *et al.* [4] present a scheme that achieves fault tolerance by horizontally and vertically shifting the configuration data. Abramovici *et al.* [5] suggest built-in self-test (BIST)-based techniques for tolerating defects that involve ‘roving self-test areas (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. Dutt *et al.* [6] require a combination of CAD tool and architecture changes to enable fast incremental routing around defects. The HP Teramac [7] 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 the defects. Xilinx’s EasyPath flow [8] enables the use of defective FPGAs, by testing if only a specific bitstream works on the FPGA candidates. However, these FPGAs can be configured only once.

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

## 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 localisation. Each of the  $n$  FPGAs in a system will have a unique defect map. 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.

Several approaches for full chip testing have been suggested in the literature. Huang *et al.* [9] and Inoue *et al.* [10] propose techniques for testing configurable logic blocks (CLBs) using integrated logic analyser techniques. Although testing CLBs is straightforward and well understood, testing interconnect has traditionally been more difficult. Renovell *et al.* [11] offer a C-testable approach to testing interconnect. Abramovici *et al.* [5] suggest 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 that uses off-the-shelf FPGAs. Neither do we have knowledge of the FPGA's layout nor 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, as each bitstream would only use a fraction of the resources of the chip.

Bitstream-dependent testing involves testing only the specific resources used by the bitstream at configuration time, rather than at manufacturing time. As 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 localisation. Xilinx's EasyPath flow [8] involves bitstream-dependent testing, but test pattern generator (TPG) and testing take several weeks, as 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 with place-and-route time, enabling a truly general purpose reconfigurable system. Das and Toubia [12] and Tahoori [13, 14] present approaches for bitstream-dependent testing of interconnect that require minimal TPG runtime. In Tahoori [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 approach for testing of delay faults. 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 possesses 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 includes a block containing several muxes that allow 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 have implemented a tool that can directly modify the bitstream of Xilinx Virtex-II Pro FPGAs for this. Fig. 1 describes an FPGA that has two ports, in which the ports can be swapped by writing into SRAM cell S0.

Fig. 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.

Fig. 3 shows an example of a three-FPGA angle system. Note that in this case, only two mappings are possible because just the leaf nodes may swap bitstreams with each other, as bitstream 0 needs to be connected to both 1 as well as 2, whereas 1 is not directly connected to 2. However, the triangle system shown in Fig. 4 allows six different mappings. Given the more favourable defect tolerant behaviour 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 favour 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, that is, defects occur at different locations in different FPGAs. We feel this condition can be sufficiently satisfied by selecting FPGAs from different lots and so on. Further we also assume that  $p$  is independent across bitstreams, that is, different bitstreams use different resources of an FPGA. This

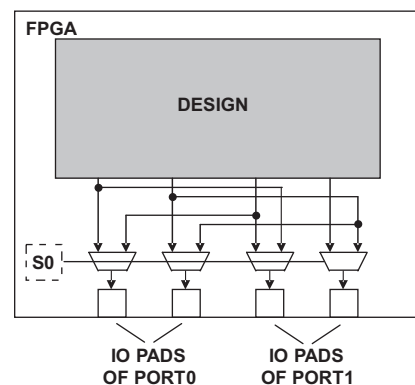


Fig. 1 Two-port FPGA

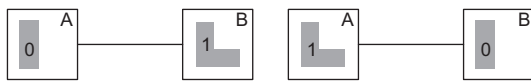


Fig. 2 Swapping with two FPGAs

condition too can be easily satisfied by restricting the nature of applications, for example, applications using cellular automata must be disallowed. Our aim is to maximise the success probability  $P$  of at least one mapping being functional. In most cases, a larger number of possible mappings imply a greater  $P$ . However, in general there is no absolute correlation between number of mappings and success probability, as 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 earlier, 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 behaviour of the system. Highly symmetric topologies can result in success probabilities approaching one 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 analyse the behaviour of topologies with good symmetry properties, namely the tree and the crossbar. Several other topologies in common use are not analysed in this paper. Examples of topologies with poor symmetry include the token ring, and meshes and tori with low dimensionality. Topologies with greater symmetry include meshes or tori 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, motherboards and switch ICs. We assume that the crossbar is implemented using some switch, the routing tables of which can be quickly changed to reflect swapping of bitstreams.

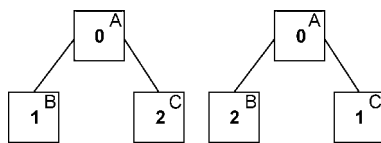


Fig. 3 Swapping with three FPGAs connected as an angle

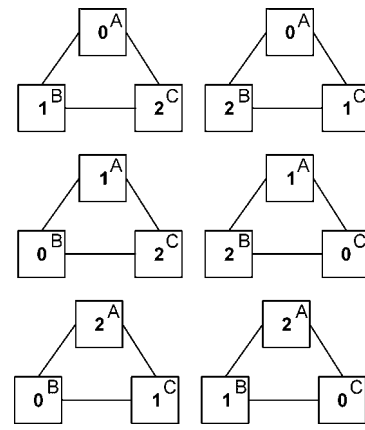


Fig. 4 Swapping with three FPGAs connected as a triangle

### 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, for this study, we also assume that communication nodes are perfect, that is, 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.

Fig. 5 indicates how symmetry can be exploited to obtain 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

Table 1: Success probabilities for three simple systems

System	$p$	$P$ without swapping	$P$ with 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

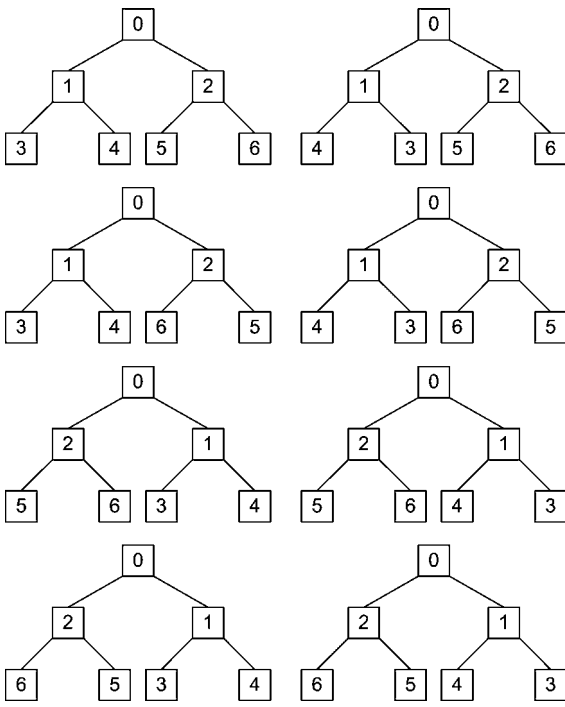


Fig. 5 Mappings in a binary tree

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 (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, for example in Fig. 5 bitstreams 3 and 4 must be no more than one hop apart

$$n_{\text{mappings\_per\_parent}} = a! \quad (1)$$

$$n_{\text{parent}} = \left[ \frac{n-1}{a-1} \right] \quad (2)$$

$$n_{\text{mappings}} = (n_{\text{mappings\_per\_parent}})^{n_{\text{parent}}} \quad (3)$$

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 (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. Dividing (4) by  $n$ , and evaluating to the limit as  $n$  approaches infinity gives the overhead per FPGA in (5). Table 2 describes this overhead for  $a$  ranging from 2 to 6. In this table, each unit refers to one of  $x^2$  units in an  $x$ -way crossbar; for example, a two-way crossbar would imply 4 units. By this analysis, there is a sweet spot at arities 3 and 4

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

$$\text{OH}_{\text{per\_FPGA}} = \frac{(a+1)^2}{a-1} \quad (5)$$

Table 2: Overhead in tree topology

Arity	Overhead (units/FPGA)		
	$d=1$	$d=2$	$d=\infty$
2	4.5	6.8	9
3	5.3	7.1	8
4	6.3	7.8	8.3
5	7.2	8.6	9
6	8.2	9.6	9.8

### 5.3 BEE2 reconfigurable supercomputer

Our research group has built a reconfigurable supercomputer called the Berkeley emulation engine II (BEE2) [15]. The current system is built from perfect FPGAs, but the architecture makes it very suitable to the defect tolerance techniques discussed in this paper. The basic structure of the system involves a set of modules, each of which are implemented as a tree of fixed  $d$  and  $a$ . 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 (6) describes the overhead of this topology

$$\begin{aligned} \text{OH}_{\text{BEE2\_xbar}} &= \left[ \frac{n}{a^d} \right]^2 \\ \text{OH}_{\text{BEE2\_tree}} &= \frac{n - \lceil n/a^d \rceil}{a-1} \cdot (a+1)^2 \\ \text{OH}_{\text{BEE2}} &= \text{OH}_{\text{BEE2\_xbar}} + \text{OH}_{\text{BEE2\_tree}} \quad (6) \end{aligned}$$

For our system, shown in Fig. 6, we chose  $a=4$  and  $d=1$ . A value of 4 was selected for  $a$ , as 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 that 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 Mapping algorithms

The problem of mapping  $n$  bitstreams on  $n$  FPGAs is identical to that of perfect matching in bipartite graphs. Although 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  $O(\sqrt{|V|} \cdot |E|)$  [16, 17].

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

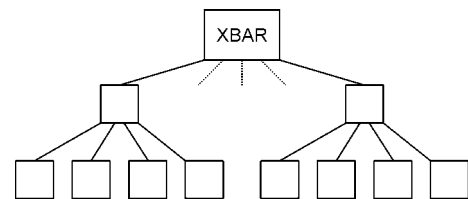


Fig. 6 BEE2 system topology

syndrome matrices of sizes  $(n/a) \times (n/a)$ ,  $(n/a^2) \times (n/a^2)$  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  $=n^2/a^2 + n^2/a^4 + \dots + 1 = (n^2/a^2) \cdot (1 - 1/n^2)/(1 - 1/a^2) = O(n^2)$ .

Fig. 7 describes an example of mapping in a tertiary tree where  $n = 9$ . The initial syndrome is a  $9 \times 9$  matrix. The first step involves nine runs of bipartite matching, each mapping 3 bitstreams on three FPGAs. A syndrome of size  $3 \times 3$  is generated such that 1 indicates success in perfect bipartite matching. A corresponding matching matrix is also generated in which 1 indicates a mapping of a bitstream on an FPGA. At the next step a  $1 \times 1$  syndrome is generated; a value of one indicates that a functional mapping exists. If a mapping exists, it can be determined by taking the AND of all the matching matrices as shown.

Table 3 describes the runtime obtained with random syndromes for determining the mapping in binary and quaternary trees. The Hopcroft–Karp algorithm was used for bipartite matching. Random syndromes were generated using  $p$ -values of 0.25, 0.5 and 0.75. The tests were performed on 1.3 GHz Itanium 2 (Madison class) machines with 4 GB of memory. The runtime is sufficiently low even for large system sizes. Also the runtime decreases with increase in arity.

## 7 Success probability

In this section, we describe the defect-tolerant performance of the different interconnection topologies. Our aim is to maximise the success probability  $P$ . A value of  $P$  close to unity implies that almost every application will be

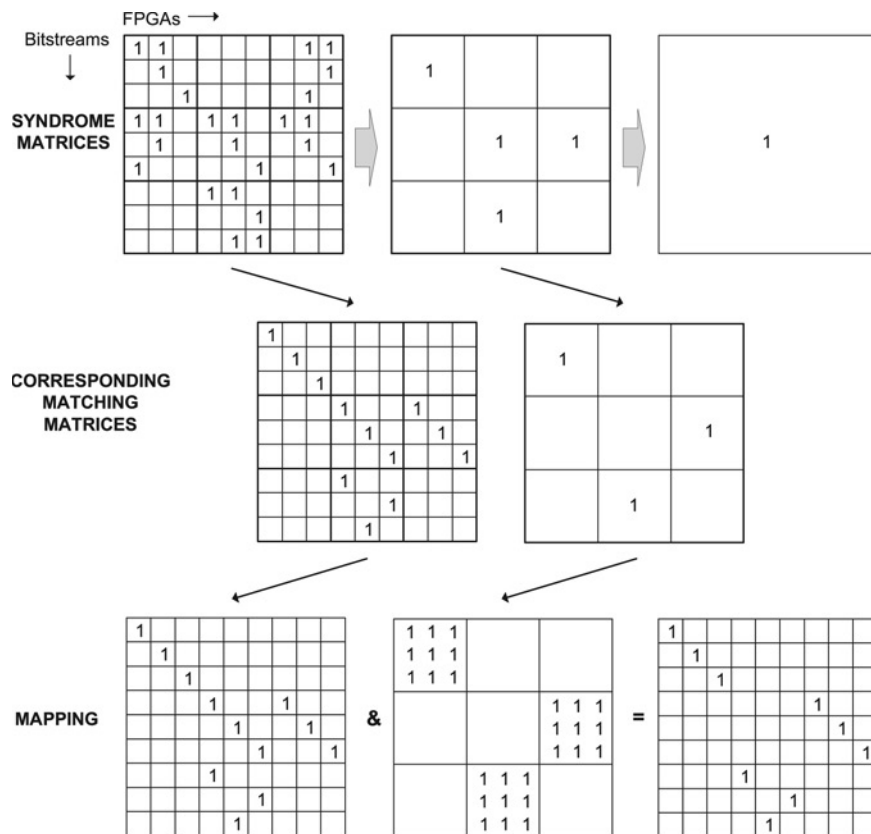
**Table 3: Mapping runtime**

System	Size $n$	Runtime, s		
		$p = 0.25$	$p = 0.5$	$p = 0.75$
Binary	64	0.035	0.045	0.051
	256	0.65	0.80	1.3
	1024	15.0	19.0	21.7
Quaternary	64	0.033	0.043	0.044
	256	0.53	0.69	0.71
	1024	8.4	10.9	11.4

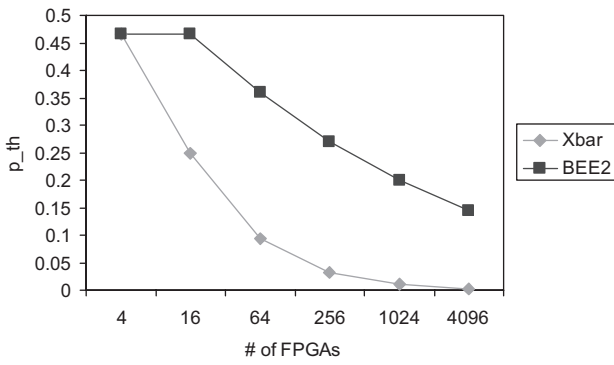
functional, and the defect-tolerant system will appear virtually the same as one built out of defect-free FPGAs. All topologies described earlier show a threshold behaviour for  $P$  with respect to base probability  $p$ .

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, which yields a perfect matching in bipartite graphs is known to be  $n \cdot \log_2 n$  [16, 17]. 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  $p_{th}$ . Fig. 8 shows this threshold behaviour 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.

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



**Fig. 7** Mapping procedure for a tertiary tree



**Fig. 8** Threshold probability for crossbar and BEE2 topologies

$p_{th}$  is sufficiently low to allow typical FPGAs to be used. A system that offers a  $p_{th}$  of 0.01 will be unnecessarily over-engineered if manufacturing never produces FPGAs with  $p$  values that are low. A higher defect density implies a lower value for  $p$ . In Section 8, 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. Fig. 9 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 quaternary tree than it is for the tertiary tree, which is lower than it is for the binary tree. We would expect this trend as an increase in the arity of the tree produces greater symmetry.

Although we have yet to precisely quantify this relationship, it seems clear that there is an inherent trade-off 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, although inexpensive to implement, lead to few opportunities to avoid defects.

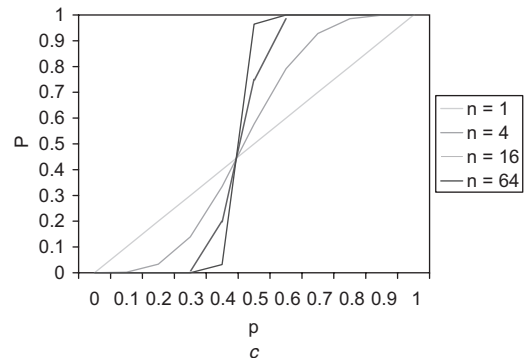
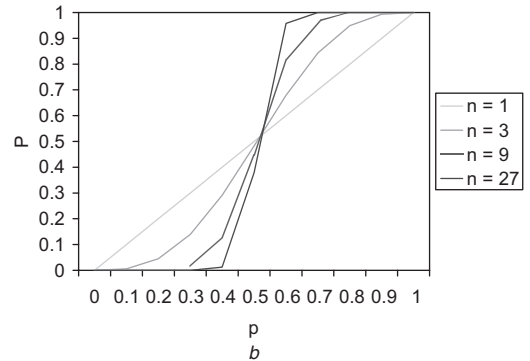
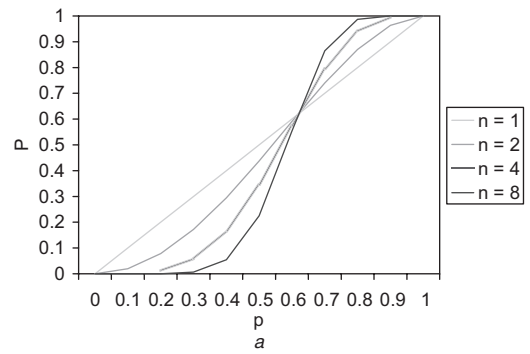
## 8 Yield analysis

In this section, we analyse the effect of yields on the performance of systems built using our approach. Although several yield models exist, we consider the classical Poisson model for yields [18]. Equation (7) 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 (8), where  $Y_C$  represents the reduction in yield due to defects in critical resources or non-repairable area and  $A_{NC}$  represents the non-critical or repairable area of the FPGA. Critical defects may include power-to-ground shorts, bad IO pads, defective configuration circuitry, and so on. 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, programmable interconnect points and so on. Defects in non-critical resources can be overcome by our defect-tolerant approach, effectively enhancing yield

$$Y = e^{-D \cdot A} \quad (7)$$

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

$$Y = Y_C \cdot Y_{NC} \quad (8)$$



**Fig. 9** Success probabilities for tree topologies

- a Binary
- b Tertiary
- c Quaternary

Any bitstream utilises only a small fraction of the FPGA's resources. Even if 70–80% of LUTs are utilised, only 5–10% of PIPs may be utilised. Because a majority of an FPGA's area is dedicated to interconnect resources to allow routability, the utilisation fraction  $U$  is quite small. Equation (9) gives the relation for  $p$ , which is identical to that of  $Y_{NC}$ , except that  $A_{NC}$  is replaced by  $A_{NC} \cdot U$ .

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

$$p = Y_{NC}^U \quad (9)$$

Fig. 10 describes the dependence of  $p$  on  $Y_{NC}$  for different utilisations. 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 almost all topologies described. In particular, the resulting value of  $p$  is satisfactory for the BEE2 supercomputer.

As all topologies described exhibit strong threshold behaviour for success probability, all FPGAs containing only non-critical defects (no critical defects) are rendered usable and appear the same as defect-free FPGAs, that is,  $Y = Y_C$ . Hence, yield improves by a factor of  $1/Y_{NC}$  using our defect tolerant approach. If the cost per chip is inversely

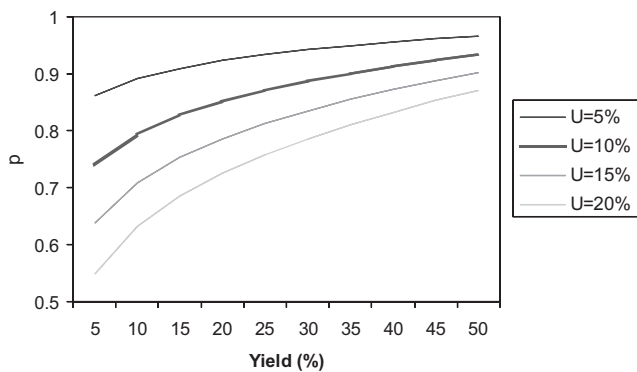


Fig. 10 Base probability as a function of yield

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, the cost of other components such as motherboards, switches and so on dominates beyond a certain point. To maximise 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.

## 9 Limitations

So far we have assumed that  $p$  is independent across different bitstreams. To ensure this property, we have limited the nature of applications. In particular, we have disallowed cellular automata-style designs, which would have an identical bitstream over all FPGAs. Here, we discuss the case where this restriction is lifted. Obviously, the success probability of mapping  $n$  identical bitstream copies on  $n$  FPGAs would be very low. Defect tolerance can still be achieved by having a much larger number of FPGAs ( $n$ ) than the number of bitstreams ( $m$ ). Equation (10) indicates the success probability as a function of  $m$ ,  $n$  and  $p$

$$P = \sum_{i=m}^n \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i} \quad (10)$$

The probability  $p$  also might not be independent across FPGAs. For instance, if all the FPGAs used belong to the same wafer lot, they may have certain common manufacturing defects. We cannot estimate how correlated the defects are, as we lack access to real industrial data. Our assumption of completely independent values of  $p$  is hence optimistic, but we hope that our approach is powerful enough to still result in enhanced effective yields, and lower overall system costs.

## 10 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 trade-off

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 realised. 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.

## 11 Acknowledgments

We would like to thank the members of the BEE2 design team for help in conceiving and developing the ideas presented in this paper. Also, many thanks to Bob Conn and Prasanna Sundararajan from Xilinx for insights and ideas about FPGA testing, and Christos Papadimitriou of UC Berkeley Computer Science for his help on the theoretical aspects of configuration swapping. This work was supported in part by the Gigascale Systems Research Center, MARCO Contract #2003-DT-660, the Berkeley Wireless Research Center, Xilinx and the California MICRO program.

## 12 References

- Hatori, F.: 'Introducing redundancy in field programmable gate arrays'. Proc. IEEE Custom Integrated Circuits Conf., 1993
- Hanchev, F., and Dutt, S.: 'Methodologies for tolerating cell and interconnect faults in FPGAs', *IEEE Trans. Comput.*, 1998, **47**, (1), pp. 15–33
- Lane, C., Zaveri, K., Yi, H., Powell, G., Leventis, P., Jefferson, D., Lewis, D., Nguyen, T., Santurkar, V., Chan, M., Lee, A., Johnson, B., and Cashman, D.: 'Programmable logic device with redundant circuitry'. US Patent 6965249, November 2005
- Doumar, A., Kaneko, S., and Ito, H.: 'Defect and fault tolerance FPGAs by shifting the configuration data'. Proc. Defect and Fault Tolerance, 1999
- Abramovici, M., Stroud, C., Hamilton, C., Wijesuriga, S., and Verma, V.: 'Using roving stars for on-line testing and diagnosis for FPGAs in fault tolerant applications'. Proc. Int. Test Conf., 1999
- Dutt, S., Shanmugavel, V., and Trimberger, S.: 'Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays'. Proc. Int. Conf. on Computer Aided Design, 1999
- Culbertson, W.B., Amerson, R., Carter, R.J., Kuekes, P., and Snider, G.: 'Defect tolerance on the teramac custom computer'. Proc. FPGAs for Custom Computing Machines, 1997
- Wells, R.W., Ling, Z., Patrie, R.D., Tong, V.L., Cho, J., and Toutouchi, S.: 'Application-specific testing methods for programmable logic devices'. US Patent 6891395, October 2004
- Huang, W.K., Zhang, M.Y., Meyer, F.J., and Lombardi, F.: 'A XOR-tree based technique for constant testability of configurable FPGAs'. Proc. Asian Test Symp., 1997
- Inoue, T., Fujiwara, H., Michinishi, H., Yokohira, T., and Okamoto, T.: 'Universal test complexity for field-programmable gate arrays'. Proc. Asian Test Symp., 1995
- Renovell, M., Portal, J.M., Figueras, J., and Zorian, Y.: 'Testing the interconnect of ram-based FPGAs', *IEEE Des. Test Comput.*, 1998, **15**, (4), pp. 89–92
- Das, D., and Touba, N.A.: 'A low cost approach for detecting, locating, and avoiding interconnect faults in FPGA-based reconfigurable systems'. Proc. IEEE Int. Conf. on VLSI Design, 1999
- Tahoori, M.B.: 'Using satisfiability in application-dependent testing of FPGA interconnects'. Proc. Design and Automation Conf., 2003
- Tahoori, M.B.: 'Application-dependent testing of FPGA interconnect'. Proc. Defect and Fault Tolerance, 2003
- Chang, C., Wawrzynek, J., and Brodersen, R.W.: 'Bee2: a high-end reconfigurable computing system', *IEEE Des. Test Comput.*, 2005, **22**, (2), pp. 114–125
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C.: 'Introduction to Algorithms' (MIT Press, 2001)
- Bollobas, B.: 'Random graphs' (Cambridge University Press, 2001)
- Cunningham, J.A.: 'The use and evaluation of yield models in integrated circuit manufacturing', *IEEE Trans. Semicond. Manuf.*, 1990, **18**, (3), pp. 422–429