Defect Tolerance at the End of the Roadmap

Mahim Mishra and Seth C. Goldstein
Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{mahim,seth}@cs.cmu.edu

Abstract

Defect tolerance will become more important as feature sizes shrink closer to single digit nanometer dimensions. This is true whether the chips are manufactured using top-down methods (e.g., photolithography) or bottom-up methods (e.g., chemically assembled electronic nanotechnology, or CAEN). In this paper, we propose a defect tolerance methodology centered around reconfigurable devices, a scalable testing method, and dynamic place-and-route. Our methodology is particularly well suited for CAEN.

1 Introduction

Photolithography techniques used today are fast approaching physical limits that will make it impossible to scale them further. New technologies that have the promise to continue the current scaling rate of device miniaturization and density growth include Next Generation Lithography (NGL) techniques such as Extreme Ultraviolet Lithography (e.g., [1, 2]), and Chemically Assembled Electronic Nanotechnology (CAEN), which uses directed self-assembly and self-alignment to construct electronic circuits out of nanometer scale devices (e.g., [3, 4, 5, 6, 7, 8]). Both these technologies hold the ultimate promise of producing feature sizes of 20 nm or less and thus extremely high device densities: current estimates show that CAEN-based devices should achieve densities of at least 10^{10} gate-equivalents/cm^2.

One significant disadvantage of both NGL and CAEN-based fabrication is that they are likely to have significantly higher defect densities than current technologies. This is especially true of CAEN-based devices: we expect that the nature of chemical fabrication will result in defect densities as high as 10% of all the fabric resources–wires, switches and logic elements. Such high defect densities require a completely new approach to manufacturing computational devices. No longer will it be possible to test a device and throw it away if it has only a handful of defects since we expect that every chip will have a significant number of defects. Instead, we must develop a method to use defective chips.

A natural solution is suggested by looking at reconfigurable fabrics, e.g., Field-Programmable Gate Arrays (FPGAs). An FPGA is an interconnected set of programmable logic elements. Both the interconnect and logic elements may be programmed, or configured, to implement any circuit. The key idea behind defect tolerance in FPGAs is that reconfigurability allows one to find the defects and then to avoid them. We expect that reconfigurable fabrics made from next generation technologies will go through a post-fabrication testing phase at which point they will be configured for self-diagnosis. Unlike the dedicated BIST structures often incorporated in current digital designs, the test circuits placed on the fabric during this self-diagnosis phase will utilize resources that will be available later for normal fabric operation, and so testing will not incur either an area or a delay penalty. The result of the test phase will be a *defect map* which contains locations of all the defects. This map can be used by place-and-route tools to layout circuits on the fabric which avoid the defects. The ability to tolerate defects in the final product eases the requirements on the manufacturing process. In some sense, this introduces a new manufacturing paradigm: one which trades-off post-fabrication programming for cost and complexity at manufacturing time.

It is worth noting here that although reconfigurable fabrics offer a solution to the high defect rates in both NGL and CAEN-based technologies, the latter are particularly well-suited for manufacturing such fabrics. Molecular junctions can be made which hold their own state, obviating the need for separate memory elements (usually SRAM cells) in silicon-based reconfigurable devices to hold the configuration state of various switch and logic elements [5, 8]. This results in a huge saving in fabric resources, and a much higher density of logic and interconnect elements.

In this paper we address the problem of finding the defects in high-density reconfigurable fabrics. The proposed approach has applicability to CAEN-based fabrics as well as deep-submicron NGL-based manufacturing techniques, if predictions of high defect rates turn out to be true. For the purposes of this paper we limit ourselves to an abstract notion of defects and manifested faults: a defect is permanent and causes the defective resource to malfunction without affecting other surrounding resources. Also, a defect always manifests itself as a fault–it is not the case that faults are manifested in some situations and not in others depending

on the operation context. Permanent stuck-open and stuck-at faults are examples of faults satisfying these conditions; in particular, we don't directly consider shorts, which may render a number of components connected to the shorted wires unusable. We also do not consider defects which do not affect component functionality but only parameters such as delay and power consumption. Although these assumptions may seem over-simplifying, they are not unrealistic for CAEN-based technologies (see [8] and references therein): the switches and logic elements will be made of molecules that should have fairly uniform operational characteristics, and most defects will occur because molecules fail to make contact or align properly during the assembly process. Also, shorts can be made very rare because for CAEN-based fabrics, we can engineer the molecules and assembly processes to be highly biased towards opens. This is likely to cause a higher overall defect rate, but a single short is more harmful than a handful of opens. Finally, we do not consider faults that may occur during the operational lifetime of the fabric, although some of our ideas for rapid defect location and reconfiguration to avoid defects have applicability for tolerating defects that occur in the field as well.

The remainder of this paper is organized as follows: Section 2 reviews previous approaches to tackle similar problems, Section 3 describes the candidate fabric architecture and how we propose to use the defect map created by the testing, Section 4 has a description of our proposed testing method, Section 5 describes our simulations to evaluate the methods we propose, and Section 6 has a brief discussion.

2 Related Work

VLSI testing is a much-studied area of research. A large number of testing strategies and design methodologies have been proposed over the years to improve the speed and accuracy of VLSI testing, and hence to enhance manufacturing yield [9]. Most such techniques have been designed around the assumption that a single, or at most very few faults exist in the portion of the circuit under test. The problem we wish to tackle is significantly harder, since a large fraction of the resources under test may be defective. One advantage we enjoy over traditional VLSI testing is that since the fabric is reconfigurable, we have the freedom to implement a circuit of choice to carry out the testing, rather than being limited to passing input vectors to the fabricated circuit.

Modern DRAM and SRAM chips and FPGAs can tolerate some defects by having redundancy built into them: for instance, a row containing a defect might be replaced with a spare row after fabrication. With CAEN fabrics, this will not be possible: it is unlikely that a portion of the fabric of any appreciable size will be defect free. Moreover, these devices are being projected as a replacement not just for memories but also for logic, where simple row-replacement-like techniques will not work since logic is less regular.

Testing and defect tolerance are widely studied problems for FPGAs and custom computing systems. A number of testing methods have been proposed for particular FPGA architectures (e.g., [10, 11, 12, 13]), as well as many FPGA architectures designed with DFT considerations in mind [14, 15]. Approaches have also been proposed for run-time defect detection and reconfiguration to avoid defects [16, 17]. In the domain of custom computing systems, the Piperench reconfigurable processor [18] and more notably the Teramac custom computer [19, 20] had a notion of testing, defect-mapping and defect-avoidance built into them. Upto 75% of the FPGAs used in the Teramac were defective; assembly was followed by a testing phase where the defects in the FPGAs were identified and mapped. Tools for generating FPGA configurations used this defect map to avoid defects. The testing strategy we propose is similar to the one used for the Teramac. However, the problem we address is significantly harder because the Teramac used CMOS devices with defect rates much lower than those predicted for next-generation technologies.

Our testing and analysis techniques have resonances with a large body of work in Statistics and Information Theory on *Group Testing* [21], which is a collection of techniques for finding members of a population which satisfy a particular property (in other words, which are "defective"). Our work is based on certain aspects of *non-adaptive, probabilistic group testing*. Different flavors of this technique have been applied to a variety of problems [22, 23, 24]. However, none of the problems discussed in the group testing literature have constraints as hard as ours: they have lower defect rates and allow a smaller granularity of access to population members than is possible here.

An alternative approach to achieve defect tolerance would be to use techniques developed for fault-tolerant circuit design (e.g., [25, 26, 27]). Such circuit designs range from simple ones involving *triple-mode redundancy* to more complex circuits that perform computation in an alternative, sparse code space, so that a certain number of errors in the output can be corrected. However, the best such techniques available today require a significant amount of extra physical resources, result in a (non-negligible) slow-down of the computation, and are hard to automate well. Also, these circuits work reliably only if the number of defects are below a certain threshold.

3 Fabric Architecture

Our defect-tolerance approach is two-fold. First, we construct a map of the defects. Then, when configuring the device to implement a particular circuit, we avoid the defects by using only the good components of the device. Our approach requires three things from a reconfigurable device:

it must be reprogrammable, it must have a rich fine-grained interconnect, and it should allow us to implement a particular logic function in many different ways. All three of these attributes are necessary for both defect detection and defect avoidance. During defect detection, we reprogram different test circuits on the device. Each different instance of a test structure gives us information about different sets of components on the device. The latter two attributes are most necessary during defect avoidance. They allow a particular circuit to be implemented without requiring us to use any of the defective components.

The particular architecture of the reconfigurable device is not essential to our defect detection or defect avoidance algorithms. In modeling the specifics of the algorithm we assume an architecture similar to an island-style FPGA, i.e., a mesh of interconnect resources surrounding islands of reconfigurable logic. The particular architecture that motivated this work is the *nanoFabric* [6, 7, 8]. The nanoFabric is an example of a possible implementation of a CAEN-based reconfigurable device. The logic blocks are cross-bars of reconfigurable diodes which can be configured to implement logic functions. Several logic blocks are grouped into clusters which can be connected using long lines that run between the clusters. Within a cluster, each logic block is connected locally to 4 neighbors. The functionality of the logic blocks and the connections to the interconnect are all reprogrammable. This architecture satisfies all three characteristics necessary for defect tolerance.

Although this paper is primarily concerned with finding the defects, we briefly address how the resulting defect map might be used to provide defect tolerance. The most obvious solution is to create a map of every defect on a particular device. Then, a compiler[1] would use the map to create a configuration for the desired circuit which avoids the defects. This is exactly how the Teramac implemented defect avoidance [19]. There are two problems with this approach. First, the final place-and-route phase could have an extremely hard time meeting timing requirements, due to pathological arrangements of defects. An even greater potential problem is that the defect map, for the large, high-density fabrics we are considering, could be as large as the fabric itself.

An alternative approach requires two levels of compilation. First, the compiler creates a *soft-configuration* which is guaranteed to be place-and-routable on any fabric as long as the defect density in any given region of the fabric is less than a certain threshold, say t. When loading onto a particular fabric, the soft-configuration is turned into a *hard-configuration* using a two-tiered defect map. The defect map would provide two-levels of detail. For a small portion of the fabric it would provide a detailed map allowing the compiler to map a circuit directly to the area in question. For the remainder of the fabric the map would provide some basic defect information about each region of the fabric. For example, the map might contain, for each cluster, the number of defects in logic and routing resources. This information would enable fast detailed-mapping when required. In addition, regions which exceed the defect-density threshold t would simply be marked as defective and not used at all. Thus, any region that is not marked defective is guaranteed not to exceed the set defect density threshold. The portion of the fabric for which an exact defect-map is available can be used for timing critical parts of the circuit, and potentially to store the second-tier defect map for the rest of the fabric. The rest of the fabric can be defect-mapped and configured as and when required by the compiler, operating in conjunction with an on-the-fly defect mapper.

[1] We use the term compiler loosely to include the CAD tool which creates the final configuration from a circuit description.

4 Proposed Testing Method

We propose configuring sets of fabric components into test circuits whose output is used to infer the defect status of individual components. Test-circuits are constructed using resources that are otherwise available for normal fabric operation and usage. Therefore, out testing method does not require any dedicated testing resources, and avoids the area and delay overhead normally associated with designs incorporating BIST.

For the description in this section, we use an abstract notion of a "defect" and a "fabric component". A defect is assumed to be permanent, and to cause some incorrectness in the output of a circuit using that defective component (see Section 1). When performing the tests on a real fabric, the test-circuits used will have to be specialized according to the type of defects being diagnosed - shorts, opens, wire-breakages etc. What we mean by a fabric component is also left unspecified. It will depend on the design of the fabric, on the granularity of reconfigurability, and on how much of the fabric resources we are willing to sacrifice to achieve quick testing. Depending on these factors, a "component" may be one or more simple logic gates, a small configurable mesh of active cross-points, or a look-up table; the on-fabric interconnect resources are also considered "components" in the sense that they are configurable and may be defective.

As an example, consider the situation in Figure 1. Five components are configured into one test-circuit, so that defects in one or more circuit components would cause the circuit output to be incorrect. By comparing the circuit's output with the correct result, it can be determined if any of the circuit's components were defective. In the first run, the circuits are configured vertically, and test circuit 2 detects a defect. In the next run, the circuits are configured horizontally, and test circuit 3 fails. Since no other errors are detected,

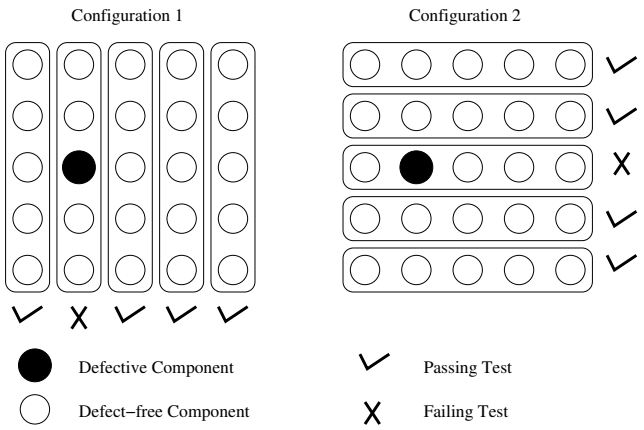

Figure 1: An example, showing how a defective component is located using two different test-circuit configurations.

we can say that the component at the intersection of these two circuits is defective, and all others are good. This testing method, used in [19], relies on the fact that most of the constructed test circuits will be defect-free. Hence, all their components can be assumed to be good. This assumption will not hold when testing CAEN-based fabrics: the defects rates will be much higher, and the test circuits will have a much larger number of components. The latter will be true for two reasons:

1. Controlling and observing a small set of resources in the interior of the fabric will require fabric interconnect resources, which may themselves be defective: an incorrect circuit result would mean a defect in the circuit's parts, or in the wires and switches used to observe the circuit's output. These interconnect resources will therefore have to be considered part of the "test circuit", thus imposing a limit on how small these circuits can be made.
2. For high-density fabrics, small test circuits would imply a long testing time, so much so that the fabrics may become economically unviable.

We believe typical test circuits used for testing these fabrics will have hundreds or even thousands of components. With a defect rate of 10%, a test-circuit with as few as a hundred components is expected to have 10 defects. This implies that the simple technique used in the example above will not work, since an overwhelming majority of circuits will contain some defects.[2] The key idea we propose to deal with this situation is that the test circuits should be more powerful: instead of simply indicating the presence or absence of defects, they should provide us more information, such as an estimate of the actual number of defective components. In particular, we have obtained encouraging simulation results using idealized test circuits that can count the actual number of defective constituent components, as well as practical LFSR-based circuits that provide approximate counts of the number of defects but still give good diagnosis results.

[2]It is easy to see analytically that for a test-circuit size of 100 and a defect rate of 10%, only about 1 circuit in 4×10^4 will be defect free. Locating a significant number of defect-free components will therefore require an enormous number of test circuit configurations.

4.1 Testing algorithm

Our testing algorithm is sketched in Figure 2. It consists of two phases: the *probability assignment* phase (lines 1–10) and the *defect location* phase (lines 11–22). The *probability assignment* phase assigns each component a probability of being defective, and discards the components which have a high probability. This should result in a large fraction of defective components being identified and eliminated from further testing. The remaining components are now likely to have a small enough defect rate that they can be tested in the *defect location* phase using the simple method employed in the example of Figure 1 to identify all the defect free components. In each phase, the fabric components are configured into test circuits in a particular orientation, or *tiling*; since each circuit uses only a small number of components, many such circuits can be configured in parallel, or *tiled*, across the fabric. For example, the circuits in Figure 1 are arranged in two tilings, vertical and horizontal. For now, we assume that arbitrary tilings are possible; this will generally not be the case in a real fabric with limited connectivity. We discuss some implications of limited fabric connectivity, and ways to deal with it, in Section 4.3.

In the *probability assignment* phase, test circuits are repeatedly configured on the fabric, and circuit results are analyzed to compute the probability of being defective for each component (*while* loop of lines 3–8). When these probabilities have stabilized, components with a high probability are discarded, or marked `suspect`. These `suspect` components are not made a part of further test-circuits. This whole process is repeated a fixed number of times (N_1, $N_1 \geq 1$) or until a point of diminishing return is reached (*for* loop of lines 2–10). Note that this phase is probabilistic, and so many good components will be misidentified as `suspect`. The purpose of running the *for* loop multiple times is to get a finer resolution of defect-probabilities for the components and thus to minimize the misidentification rate. The remaining points of interest here are the *type-1* test-circuits used in line 5, and the method for computing defect probabilities in line 6. These are described in greater detail in Sections 4.2 and 4.3, respectively.

After the *probability assignment* phase, a number of components have been marked `suspect` and are not included in the further test-circuits. For the remaining components, the defect density is expected to be low enough that a substantial number of test-circuits will be defect-free. These defect-free components are identified in the *defect-location* phase (lines 11–22). In this phase, the circuits used (*type-2 test circuits*,

```
    // Probability Assignment Phase
1   mark all fabric components not suspect
2   for iteration from 1 to N1 do
3       while probabilities not stable do
4           for all fabric components marked not suspect do
5               configure components into type-1 test-circuits using a particular tiling
6               compute defect probability for each component using circuit results from current iteration
7           done
8       done
9       mark components with high defect probability as suspect
10  done

    // Defect Location phase
11  for iteration from 1 to N2 do
12      while results improve do
13          for all fabric components marked not suspect or not defective do
14              configure components into type-2 test-circuits using a particular tiling
15              for all circuits with correct output do
16                  mark all circuit components not defective
17              done
18          done
19      done
20      mark some suspect components not suspect
21  done

22  Mark all remaining components as defective
```

Figure 2: Proposed testing algorithm.

line 14) return a wrong answer in the presence of a defect, but don't need to provide any more information. Components of error-free circuits are marked `not defective`. This whole process is repeated till no more good components are identified (*while* loop, lines 11–19). At this point, some of the components previously marked `suspect` are added back (line 20), and the whole process is repeated; this is done a total of N_2 times. The purpose of this is to try to reduce the number of components misidentified as `suspect` by the *probability-assignment* phase.

We define a quality metric, *recovery*, to judge our algorithm. Recovery is the percentage of defect-free components which our algorithm identifies as such. For example, if the fabric has a 10% defect rate, and the algorithm identifies 45% of the components as defect free, the recovery is 50% (45/(100-10)$\times$100%). The recovery value is usually less than 100% because the algorithm has many false positives, i.e., good components are identified as bad. In general, recovery will depend on the type of test-circuits used, number of tests run and the rigor of the post-testing analysis. There are a number of trade-offs that can be made between testing time and recovery, for example by adjusting the value of N_1 and N_2, or by changing the termination condition for the *while* loops on lines 3 and 12. In particular, the value of N_2 has been set to more than 1 in our simulations and the step on line 20 has been added in an effort to maximize recovery. It should be noted that if the test circuits used in the *defect location* phase can detect all modeled defects, this method of defect-mapping will never produce false negatives (i.e., bad components which are identified as good): all the components that the algorithm says are good will actually be good.

Another important trade-off in the method is the obliviousness or non-adaptiveness of the test-circuit generation. By this, we mean that the results of previous tests are not used to generate new circuits. In the algorithm as it stands, a small amount of re-routing of test circuits is required after each iteration of the *for* loops in the two phases, when some components are discarded or added back. This re-routing is unlikely to require much effort since only routes to bypass certain components need to be included in pre-computed tile configurations; even this small amount of routing effort can be traded-off against recovery by adjusting N_1 and N_2.

4.2 Some candidate test-circuits

We need test-circuits that can give us some notion of the number of defects in the circuit's constituent components.

We have considered two kinds of circuits for this purpose: idealized *counter* circuits that can actually count the number of defects, and *none-some-many* circuits, which can tell us, with reasonable certainty, if the circuit had none, some or many defective components.

Counter circuits: These are idealized circuits that can count the number of defects, upto a certain threshold. For example, if the circuit's threshold is t, it can tell us if there are 0, 1, 2, ..., t or $> t$ defects in the circuit's components. Naturally, circuits with a higher threshold are more powerful and give better recovery results than those with a lower threshold. Although it is extremely difficult to practically realize such counter circuits with high thresholds, our simulation results in Section 5 show that even circuits with a threshold of 1 (i.e., which can only tell us if there were 0, 1 or more defects) can give significant recovery for moderate defect rates. For certain defect types, designing such low-threshold counter circuits may be possible.

None-some-many circuits: These circuits are a weaker version of the counter circuits described above. They can tell us, with some degree of accuracy, if the circuit contained *none*, *some* or *many* defects. We have designed simple LFSR-based circuits that can give us such information. These test circuits operate as follows:

- A set of components are organized into an LFSR, which is provided with an initial input and run autonomously for a while. Its signature is then matched against the correct output. If the signature matches, the circuit is assumed to contain no defects. Note that this does not mean the circuit is defect-free, because a correct output may have been obtained because of aliasing.
- If the large LFSR produced a signature mismatch, it is split into some number of smaller LFSRs (say 4), which are again provided initial inputs and run autonomously. If less than half of the smaller LFSRs have a signature mismatch, the entire circuit is assumed to contain *some* defects; if there are more mismatches, the circuit is assumed to contain *many* defects.

Breaking up a larger LFSR into some smaller ones should require minimal reconfiguration, since only a small number of forward and feedback connections need to be redirected. We have found that the two-step schema above works better than using smaller LFSRs from the start because smaller LFSRs have a higher aliasing probability. Also, in the initial stages of testing, most circuits will be found to contain some or many defects. However, as components are marked `suspect` and removed from subsequent testing steps, many of the large LFSRs will turn out to be defect-free, obviating the need to run the smaller LFSR circuits. This should speed up the testing. An alternative to LFSRs would be circuits based on linear cellular automata [28], which have the advantage of not having long feedback connections. However, low aliasing probabilities are harder to achieve with cellular automata and they are also harder to design since they need more fabric resources.

4.3 Analysing circuit outputs

Once the results of the test circuits have been obtained, they are used to determine the probability of each individual component being defective. We have considered two methods for doing this analysis:

"Sorting" analysis: Let a component c be part of n different circuits. Based on the results of these n circuits, we calculate a *fault-value* for the component, as follows: if the test circuits are *counter* circuits, the fault-value is simply the sum of the number of defects in each of the n circuits. If the circuit is an LFSR-based *none-some-many* circuit, we assign numerical weights to each result (e.g., 2 to *many* defects, 1 to *some* and 0 to *none*) and sum up all n weights for the component. Once this calculation has been performed for all components under test, they are sorted according to their fault-values and components with higher fault-values are assigned a higher probability of being defective. This method involves simple calculations and places no specific restrictions on the shape or nature of the tilings.

Bayesian analysis: Again, let a component c be a part of n different test circuits. Let p be the *a priori* known defect rate in the fabric, obtained through some initial testing or from knowledge of the manufacturing process. Let a_1, a_2, ..., a_n represent numerical results for each of these circuits (these can be actual defect counts for *counter* circuits, or numerical weights for the *none-some-many* circuits as described above). We need to find the posterior probability of component c being defective given our knowledge of the circuit results. Let A be the event that c is good, and let B be the event of obtaining the circuit results that we have obtained for the n circuits. Therefore, we need to find $P(A|B)$.

Now, from Bayes' rule,

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A \cap B)}{P(A \cap B) + P(\overline{A} \cap B)}$$

If c is the only component that the n circuits share, this equation simplifies to the following:

$$P(A|B) = \frac{1}{1 + \frac{(1-p)^{n-1} n^n}{p^{n-1}(n-a_1)(n-a_2)...(n-a_n)}}$$

This equation is solved for each component to obtain its probability of being good (the probability of being bad, which is required by the algorithm, is simply this value subtracted from 1). The simple closed-form expression obtained above holds true if all the circuits that a component is

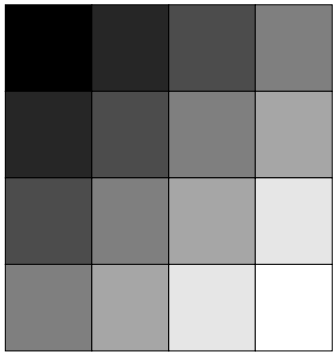

Figure 3: A schematic representation of how testing will proceed in a wave-like manner through the fabric. The black area is tested and configured as a tester by the external tester; each darker-shaded area then tests and configures a lighter-shaded neighbor.

part of have only that component in common. It can easily be shown that in general, the amount of computation grows exponentially with the number of components that two different circuits can share. This is a significant limitation of the method: on a fabric with limited routing resources, it severely restricts the number and type of tilings that are practically realizable. Although the sorting analysis gives results inferior to the Bayesian analysis, it does not suffer from this limitation. We are currently exploring ways to combine these two techniques to get around this while maintaining high recovery.

4.4 Scaling the testing process with fabric size

A short testing time is crucial for the usability and low cost of these fabrics, so it is important to ensure that the testing procedure scales with fabric size. We shall begin by analyzing the testing strategy above to see how long it takes to run.

The size of a test-circuit will depend on the granularity of access the fabric provides us - in general, smaller circuits provide more accurate information but are harder to realize. Let the circuit size be k. Then, since the circuits in a tiling are independent of each other, a $k \times k$ piece of fabric can be configured to run k test circuits in parallel. Let the average number of iterations of the two *while* loops in lines 3–8 and 12–19 of the algorithm be x and y respectively (x and y will depend on the termination condition used for the two loops). Then, the total number of tilings used equals $N_1x + N_2y$. We have observed empirically that if d is the defect rate, the number of tilings required before recovery stops improving scales as $O(k \times d)$. Therefore, for testing $k \times k$ components, we require $O(k \times d)$ fabric reconfigurations. If the fabric is larger then $k \times k$ it can be split into many sections of size $k \times k$, each of which can be tested separately.

We envisage that the reconfigurability of the fabric can be leveraged to reduce the time spent on an external tester significantly. Once a part of the fabric is tested and defect-mapped, it can be configured to act as a tester for the other parts. Also, there is nothing to prevent us from having multiple testers active simultaneously. In such a scenario, the first area to be tested tests its adjacent ones, which test their adjacent ones and so on, and the testing can move in a wave through the fabric (see Figure 3). For large fabrics, multiple such waves may grow out from different externally-tested areas. Now, as the fabric size increases, testing time grows linearly with the distance this wave has to traverse through the fabric, which is proportional to the length of the fabric's edge, and to the square root of the number of components in the fabric.

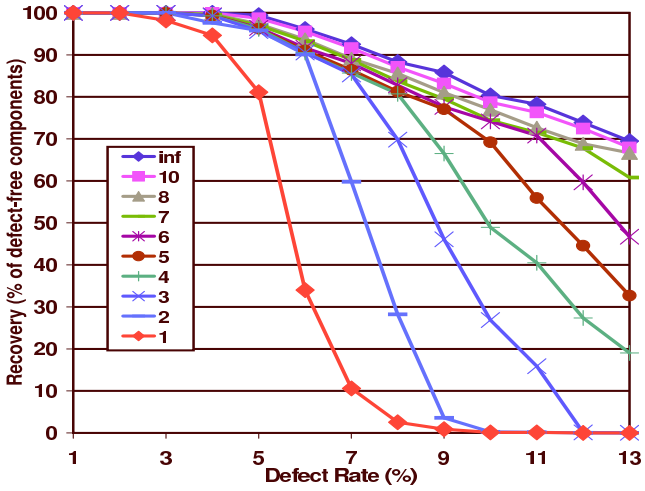

Figure 4: Recovery vs. defect rate for *counter* circuits using *sorting* analysis of circuit results. Each line represents a circuit with a different upper bound for the number of defects that can be counted. *inf* stands for infinity.

5 Evaluation

We performed simulations of our algorithm to determine its efficacy and to evaluate the two types of test-circuits and analysis methods described in Section 4. These simulations were carried out using a very abstract notion of the fabric architecture and defect model: the fabric was assumed to consist of a large number of "components" arranged in a rectangular array with practically unlimited routing resources to connect them together. This allowed us to configure arbitrary test-circuit tilings onto the fabric, which will in general not be feasible on a real fabric. It was also assumed that suitable test-circuits were available to identify each of the different types of defects that can occur on the fabric. Although this abstracts away all the details of the fabric architecture and device failure model, our results are still a fairly good indication of the level of recovery achievable in such a high-defect-rate regime.

The simulations were carried out using test-circuits that had about 100 components each, and with fabrics that had defect

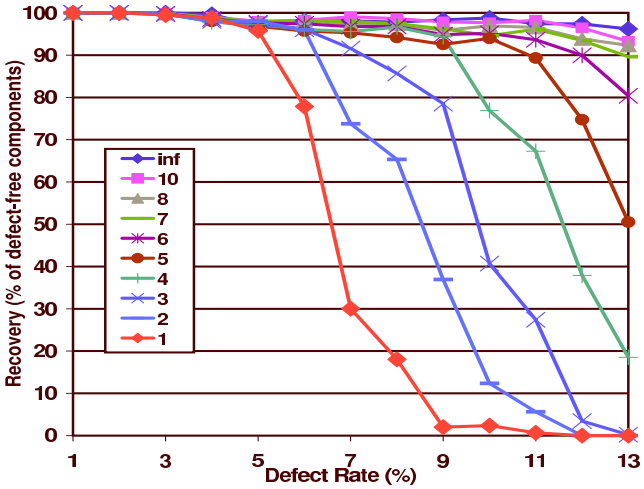

Figure 5: Recovery vs. defect rate for *counter* circuits using *Bayesian* analysis of circuit results. Each line represents a circuit with a different upper bound for the number of defects that can be counted. *inf* stands for infinity.

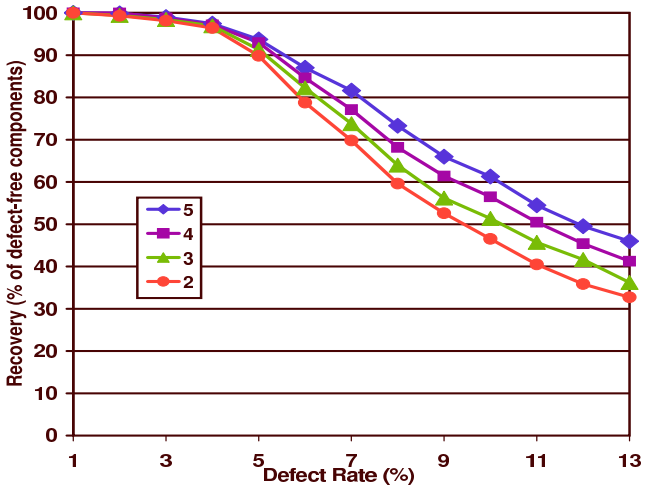

Figure 6: Recovery vs. defect rate for *none-some-many* circuits using *sorting* analysis of circuit results. The label for each curve represents the number of smaller LFSRs that the larger LFSR is split into.

rates ranging from 1 to 13 percent. What this actually means is that our analysis methods were tested for an average of 1 to 13 defects per test circuit, and our results are valid if test circuits used on actual fabrics have a defect count approximately in this range. Therefore, the size of the test-circuit will have to be adjusted according to the defect rate of the fabric: fabrics with a small defect rate can be tested using larger test circuits, but fabrics with higher defect rates will require smaller circuits and therefore the architecture will need to provide more fine-grained access to fabric internals.

Figures 4 and 5 present simulation results for *counter* circuits using, respectively, the *sorting* and *Bayesian* analysis. Figures 6 and 7 present results for the LFSR-based *none-some-many* circuits. For all of the above simulations, we assume that the defects have a random, uniform distribution throughout the fabric. Experience with VLSI fabrication has shown that defects are often clustered rather than uniformly scattered. However, the assumption of randomly scattered defects is pessimistic compared to clustered defects: if the defects are clustered, a larger number of circuits are expected to be defect-free, and hence defect diagnosis can be expected to be easier, as we show later in this section.

The *counter* circuit simulations were carried out using a number of test circuits with different thresholds on the number of defects they can count. These thresholds were varied from 1 (the circuit can only tell us if there were 0, 1 or more defects) to 10. For comparison, a circuit that can count an unbounded number of defects was also included. The simulated circuits were also allowed to return an incorrect result with a small probability. The test cycle was carried on till the results stopped improving, i.e., the precomputed tilings failed to identify any new defect-free components. The number of tilings required for each circuit varied from about 20 to 200, depending on the type of circuit, the fabric defect rate and which type of analysis was being used (sorting analysis needed about twice the number of tilings needed by the Bayesian analysis). Such a large number of tilings is possible under our unrestricted fabric connectivity assumption, but may not be achievable with a real fabric. For both types of analysis, good recovery results were achieved with circuits having relatively low counting thresholds. The results indicate that significant recovery is achievable if the test-circuit can count about a third of the expected number of defects per circuit (e.g., a circuit that can count 3 or 4 defects gives good results if the average number of defects per circuit is around 10).

For the *none-some-many* circuits, LFSRs were first configured using about a hundred components, and these were then broken up into 2, 3, 4 or 5 smaller LFSRs (recall that the results of the large LFSR as well as each of the smaller pieces are taken into account while deciding if the circuit has none, some or many defects). Breaking into more pieces gives more accurate information about the components being tested, but requires more effort to get inputs to and outputs from the circuits. The simulated circuits produced incorrect results with the expected aliasing probability for circuits of that size. These circuits required about 50 to 100 tilings, before results stopped showing an improvement. The number of tilings required depended in the expected way on the type of circuit used, fabric defect rate and analysis technique.

To show that the results presented so far will only improve in the presence of clustered defects, we generated fabrics in which the defects occurred in clusters, with components around the center of the cluster having a normal probability

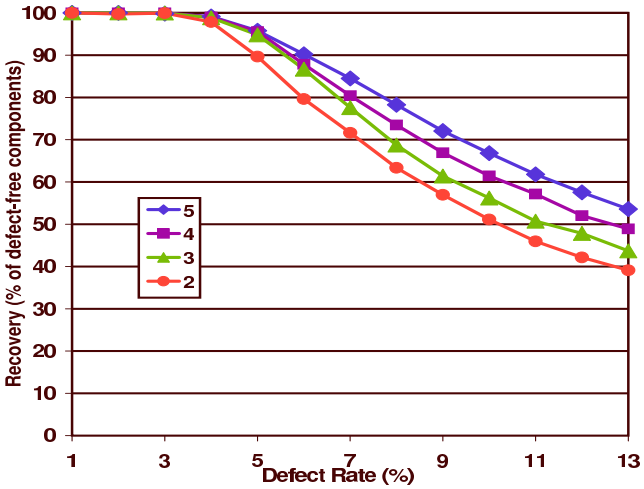

Figure 7: Recovery vs. defect rate for *none-some-many* circuits using *Bayesian* analysis of circuit results. The label for each curve represents the number of smaller LFSRs that the larger LFSR is split into.

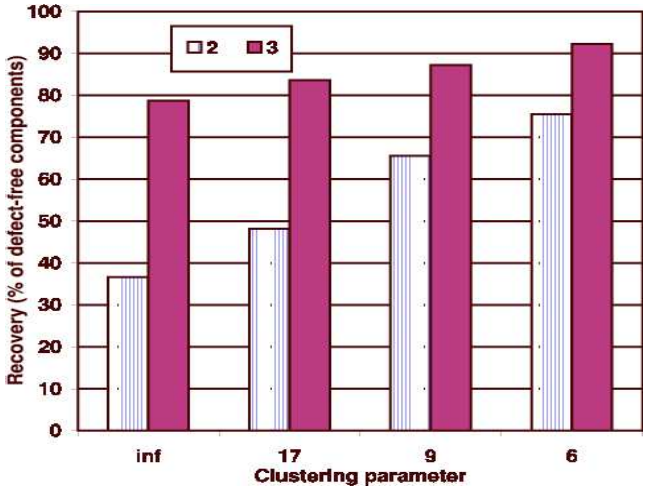

Figure 8: Recovery results for clustered defects, using counter circuits of thresholds 2 and 3 and Bayesian analysis. The horizontal axis shows different values of a clustering parameter, which corresponds to the tightness of the clusters. Larger values represent looser clusters: infinity (*inf*) represents unclustered, uniformly distributed defects, or the results in Figure 5.

distribution of being defective. Clusters of different tightness were obtained by varying the standard deviation of the normal distribution. These results are presented in Figure 8: all the fabrics had a defect rate of 9%; the bars to the left represent larger standard deviations and hence larger clusters, while those to the right represent tighter clusters. The leftmost bars correspond to a standard deviation of infinity, which is essentially the uniform distribution. We simulated counter circuits with thresholds of 2 and 3 and Bayesian analysis. As the clusters get tighter, the recovery results for the circuits with threshold 2 improve dramatically. For the threshold 3 counter, the results were good to start with, but still show a small improvement.

To summarize, the more sophisticated Bayesian analysis has significantly better recovery than the simpler sorting analysis. Also, simulations using the sorting analysis technique needed about twice the number of reconfigurations needed by the Bayesian analysis. However, the sorting analysis is much easier to implement, particularly because it places far fewer restrictions on the types of tilings that can be used (recall that the Bayesian analysis required that any two test circuits have at most one component in common). We are presently investigating other methods of analysing circuit results, including a combination of sorting and Bayesian analysis, to reduce the number of reconfigurations required as much as possible while not sacrificing recovery.

6 Discussion

Next generation manufacturing technologies are expected to achieve extremely high device densities, yielding computational fabrics with many billions of components. However, this boon comes at the cost of large defect densities. In order to make the entire process economical it is important that the resulting fabrics be defect tolerant. Assuming one can find the defects, one possible defect tolerant architecture is a reconfigurable computing device. In this paper we have shown that it is possible to find the defects in a reconfigurable computing device, even when the device is large and has many defects. Our method uses test-circuits that provide more information about the defect status of their components than simply the presence or absence of defects. We have presented a testing algorithm that uses such test circuits to obtain a reasonably high level of recovery of fabric components.

Our algorithm attempts to minimize the amount of rerouting required at test time, since rerouting is a slow and computationally expensive procedure. Therefore, we restrict ourselves to using pre-computed tiling configurations which undergo a minimal amount of adaptation at test time. Of course, once the algorithm has identified some fabric resources as being defect free, test circuits can be generated which incrementally test fabric resources whose defect status is unknown, potentially achieving close to 100% recovery. A naive example of such an *adaptive* testing method would be one that tests each component by making it part of a test-circuit all of whose other components are known to be defect-free. Although this method does the required job, it is very inefficient, and identifying an efficient way to carry out this final testing is a focus of our current work.

The final circuit configuration obtained after compilation will depend on the defect distribution, and will therefore be different for each individual fabric. With such a scheme, it will be difficult to obtain hard guarantees for the delay of placed and routed circuits. We envision that the compiler

will generate locally synchronous, globally asynchronous circuits, with very loose timing constraints. This means that timing constraints will have to be met only for small parts of the circuit that can be placed within a small, localized area. Communication with more distant parts of the circuit takes place through asynchronous signals for which timing and delay is not so much an issue. Using asynchronous logic will also give us some protection against defects which may only affect component delays and not functionality

Acknowledgements

This research is funded in part by the National Science Foundation under Grant No. CCR-9876248 and by Darpa under contract #MDA972-01-03-0005. The authors would like to thank Shawn Blanton and the anonymous reviewers for their helpful comments and suggestions.

References

[1] K. Diefenderhoff, "Extreme lithography," *Micrprocessor Report*, no. 6/19/00-01, June 2000.

[2] "International technology roadmap for semiconductors 2001: Lithography," International Sematech, available online from http://public.itrs.net.

[3] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K.-H. Kim, and C. M. Lieber, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, pp. 1313–1317, Nov. 2001.

[4] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, pp. 1317–1320, Nov. 2001.

[5] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath, "Electronically configurable molecular-based logic gates," *Science*, vol. 285, pp. 391–394, July 1999.

[6] S. C. Goldstein and M. Budiu, "Nanofabrics: Spatial computing using molecular electronics," in *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA 2001)*, July 2001, pp. 178–191.

[7] S. C. Goldstein and D. Rosewater, "Digital logic using molecular electronics," in *Proccedings of the 2002 IEEE International Solid-State Circuits Conference (ISSCC '02)*, San Francisco, CA, Feb. 2002, pp. 204–205.

[8] S. C. Goldstein and M. Budiu, "Molecules, gates, circuits, computers," in *Molecular Nanoelectronics*, M. A. Reed and T. Lee, Eds. 25650 North Lewis Way, Stevenson Ranch, California 91381- 1439, USA: American Scientific Publishers, May 2003.

[9] V. D. Agrawal and S. C. Seth, *Tutorial: Test Generation for VLSI Chips*. 1730 Massachusetts Avenue, N.W., Washington, DC 20036-1903: Computer Society Press, 1988.

[10] T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal fault diagnosis for lookup table FPGAs," *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 39–44, January-March 1998.

[11] A. Doumar, S. Kaneko, and H. Ito, "Defect and fault tolerance FPGAs by in shifting the configuration data," in *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, Albuquerque, NM, USA, November 1-3 1999, pp. 377–385.

[12] W. K. Huang, X. T. Chen, and F. Lombardi, "On the diagnosis of programmable interconnect systems: Theory and application," in *Proceedings of the 14th VLSI Test Symposium*, April 28-May 1 1996, pp. 204–209.

[13] Y. Yu, J. Xu, W. K. Huang, and F. Lombardi, "Minimizing the number of programming steps for diagnosis of interconnect faults in FPGAs," in *Proceedings of the 8th Asian Test Symposium*, Shanghai, China, Nov. 1999, pp. 357–362.

[14] X. Chen, W. Huang, F. Lombardi, and X. Sun, "A row-based FPGA for single and multiple stuck-at fault detection," in *Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, Lafayette, LA, November 13-15 1995, pp. 225–233.

[15] P. Lala, A. Singh, and A. Walker, "A CMOS-based logic cell for the implementation of self-checking FPGAs," in *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, Albuquerque, NM, USA, November 1-3 1999, pp. 238–246.

[16] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, vol. 6, no. 2, pp. 212–221, June 1998.

[17] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic fault tolerance in FPGAs via partial reconfiguration," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000)*, Napa Valley, CA, Apr. 2000, pp. 165–174.

[18] S. K. Sinha, P. M. Karmachik, and S. C. Goldstein, "Tunable fault tolerance for runtime reconfigurable architectures," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000)*, Napa Valley, CA, Apr. 2000, pp. 185–192.

[19] B. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider, "Defect tolerance on the Teramac custom computer," in *Proceedings of the 1997 IEEE Symposium on FPGA's for Custom Computing Machines (FCCM '97)*, Napa Valley, CA, April 16-18 1997.

[20] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716–1721, 12 June 1998.

[21] D.-Z. Du and F. K. Hwang, *Combinatorial Group Testing and its Applications*, 2nd ed., ser. Series on Applied Mathematics, F. Hwang, Ed. New York: World Scientific, 2000, vol. 12.

[22] R. Dorfman, "The detection of defective members of large populations," *Annals of Mathematical Statistics*, vol. 14, pp. 436–440, Dec. 1943.

[23] J. K. Wolf, "Born again group testing: Multiaccess communications," *IEEE Transactions on Information Theory*, vol. IT-31, no. 2, pp. 185–191, Mar. 1985.

[24] E. Knill, W. J. Bruno, and D. C. Torney, "Non-adaptive group testing in the presence of errors," Los Alamos National Laboratory, Los Alamos, NM, Tech. Rep. LAUR-95-2040, Sept. 1996.

[25] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems*, 2nd ed. Burlington, MA: Digital Press, 1992.

[26] N. Pippenger, "Developments in "The synthesis of reliable organisms from unreliable components"," *Proceedings of Symposia in Pure Mathematics*, vol. 50, pp. 311–324, 1990.

[27] D. A. Spielman, "Highly fault-tolerant parallel computation," in *Proceedings of the 37th Annual IEEE Conference on Foundations of Computer Science (FOCS'96)*, Burlington, VA, USA, Oct. 14–16 1996, pp. 154–163.

[28] M. Serra, T. Slater, J. C. Munzio, and D. M. Miller, "The analysis of one-dimensional linear cellular automata and their aliasing properties," *IEEE Transactions on Computer Aided Design*, vol. 9, no. 7, pp. 767–778, July 1990.