Scalable Defect Tolerance for Molecular Electronics

Mahim Mishra and Seth C. Goldstein
{mahim,seth}@cs.cmu.edu

Computer Science Department
School of Computer Science
Carnegie Mellon University

Abstract

Chemically assembled electronic nanotechnology (CAEN) is a promising alternative to CMOS-based computing. However, CAEN-based circuits are expected to have huge defect densities. To solve this problem CAEN can be used to build reconfigurable fabrics which, assuming the defects can be found, are inherently defect tolerant. In this paper, we propose a scalable testing methodology for finding defects in reconfigurable devices.

1 Introduction

One alternative to CMOS-based computing that is under intense investigation is chemically assembled electronic nanotechnology (CAEN). CAEN uses directed self-assembly and self-alignment to construct electronic circuits out of nanometer scale devices (e.g., [1, 2, 3]). Current estimates show that CAEN-based devices should achieve densities of at least 10^{10} gate-equivalents/cm^2. This represents a significant increase in resources over state-of-the art photo-lithography based silicon. CAEN-based devices will, however, suffer from a major disadvantage vis-a-vis CMOS devices: their defect densities will be significantly higher than for CMOS-based devices. In fact, we expect that the very nature of CAEN-based fabrication will result in defect densities of as much as 10%. 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 a defect since we expect that every chip will have a significant number of defects. Instead, we will have to devise a way to use defective chips.

A natural solution is suggested by looking at reconfigurable fabrics, i.e., Field-programmable gate arrays (FPGAs), and the work on the Teramac custom computer [4, 5]. 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 Teramac is essentially a very large FPGA with a very rich interconnect that works in spite of the fact that 75% of the chips contained in the Teramac had some number of defects. The key idea behind making Teramac work is that reconfigurability allows one to find the defects and then to avoid them. Before the Teramac can be used it is first configured for self-diagnosis. The result of the diagnosis phase is a map of all the defects. Then, one implements a particular circuit by configuring around the defects.

In some sense, Teramac introduces a new manufacturing paradigm; one which trades-off complexity at manufacturing time with post-fabrication programming. The reduction in manufacturing time complexity makes reconfigurable fabrics a particularly attractive architecture for CAEN-based circuits, since directed self-assembly will most easily result in highly regular, homogeneous structures. We will call CAEN-based reconfigurable fabrics, nanoFabrics [3]. We expect that the fabrication process for these fabrics will be followed by a testing phase, where a defect map will be created and shipped with the fabric. The defect map will be used by compilers to route around the defects.

In this paper we address the problem of finding the defects in a nanoFabric. This problem would be trivial if we could test the individual components. However, this will not be possible. In general, the testing strategy should needs to satisfy the following constraints:

- It should not require access to the individual components.
- It should scale with the number of defects.
- It should scale with fabric size, so that testing does not become a bottleneck in the manufacturing process.

The remainder of this paper explores some ways of solving this problem: Section 2 describes previous approaches to tackle similar problems, Section 3 describes the requirements that a testing strategy will need to satisfy and presents our proposed testing strategy, Section 4 describes simulations we have carried out to validate the methods we propose, Section 5 outlines some of the work that remains

to be done, and Section 6 presents our conclusions.

2 Related Work

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

Problems similar to this have been addressed in the domain of custom computing systems. For example, the Piperench reconfigurable processor [6] and more notably the Teramac custom computer [4, 5] 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. Compilers for generating FPGA configurations then use this defect map to avoid these defects. The testing strategy we are proposing is similar to the one used for the Teramac. However, the problem we address is significantly harder because the Teramac used CMOS devices whose defect rates are much lower than those predicted for nanofabrics.

An alternative approach to achieve defect tolerance would be to use techniques developed for fault-tolerant circuit design (e.g., [7, 8]). 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 (up to half the minimum distance between any two code words) can be corrected. Fault-tolerant circuits do not suit our purpose for the following reasons:

1. The best techniques for fault tolerance available today require a significant amount of extra physical resources, and also result in a (non-negligible) slowdown of the computation.

2. These circuits work reliably only if the number of defects are below a certain hard threshold.

3. Designing such circuits is a non-trivial task, particularly for a compiler.

3 Testing Strategy

As was done on the Teramac, we propose configuring the components on the nanofabric[1] into test circuits, which are capable of giving us information about the presence or absence of defects in their constituent components. Each component is made a part of many different test circuits, and information about the error status of each of those circuits is collected. This information is used to deduce and confirm the exact location of the defects.

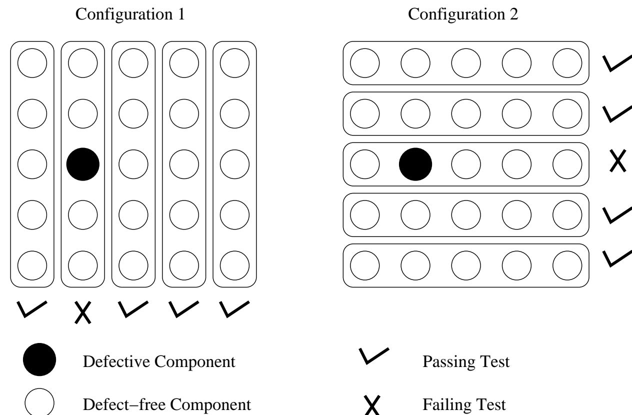

Figure 1: *An example, showing how a defective component is located using two different test-circuit configurations. The components within one rectangular block are part of one test-circuit*

As an example, consider the situation in Figure 1. Five components are configured into one test-circuit, which computes a simple mathematical function. This function is such that defects in one or more circuit components would cause the answer to diverge from the correct value; therefore, by comparing the circuit's output with the correct answer, the presence or absence of any defects in the circuit components can be detected. In the first run, the components are configured vertically, and test circuit 2 detects an defect. In the next run, the components are configured horizontally, and test circuit 3 fails. Since no other errors are detected, we can say that the component at the intersection of these two circuits is defective, and all others are good.

For the rest of this paper, we use the following terminology, and make the following assumptions about the fabrics and the nature of the defects:

1. The fabric under test has n components.

[1] We are deliberately leaving the meaning of "component" unspecified. It will depend on the final design of the nanofabric: a component may be one or more simple logic gates, or a look-up table implementing an arbitrary logic function; also, the on-fabric interconnects will also be "components" in the sense that they may also be defective.

2. Defects are randomly occurring, independent events with each component having a probability p of being defective.

3. The total number of defects in the fabric is m (and, with a large enough fabric, this is expected to equal $n * p$).

4. Each test-circuit we use has size k; we assume that we have an arbitrary level of connectivity on the fabric, and a test-circuit can consist of any set of components and not just ones that are adjacent to one another [2].

5. The components on the fabric are configured as part of many different test-circuits, and we call each of these configurations a *tiling* (in the simple example above, we used 2 tilings: horizontal and vertical). We assume that circuits within the same tiling do not share any components, and that any 2 circuits belonging, respectively, to 2 different tilings share at most one component. We make this assumption to simplify the analysis in the rest of this paper; testers that share more than one component are certainly possible but the results would be more difficult to analyze. We let t denote the number of tilings we make.

As has been explained in Section 1, we do not have access to the individual fabric components. This implies that our test circuits will have to be large, and will consist of tens and perhaps even hundreds of components. With the high defect rates, each circuit can now potentially have multiple defective components; this will considerably complicate the simple picture presented in the example above. In the rest of this section, we will show how the testing strategy will need to change as we try to scale with defect density and with the fabric size.

3.1 Scaling with Defect Density

We shall begin by analyzing some simple cases where the defect densities are low: these will help explain our approach, and also motivate our proposal to deal with higher defect densities. Using the terminology described above, $k * p$ is the number of defective components that each test circuit is expected to have. Not surprisingly, the difficulty of locating the errors scales with $k*p$; we analyze the problem for the following three cases and describe how our tests can handle each of them:

1. $k * p \ll 1$
   e.g., $k = 10, p = 0.01, k * p = 0.1$

2. $k * p \approx 1$
   e.g., $k = 10, p = 0.1, k * p = 1$

3. $k * p \gg 1$
   e.g., $k = 100, p = 0.1, k * p = 10$

3.1.1 $k * p \ll 1$

The defect rates here are small, and there is very little likelihood of a test circuit containing a defect. For example, with $k = 10$ and $p = 0.01$, 9 out of 10 test circuits are expected to be free of defects. The defect rates found in today's CMOS devices would put them in this regime. The testing strategy for such defect rates would consist of the following steps [3]:

1. Using a particular tiling, configure the components into test circuits which indicate the presence or absence of an error. If there is no error, mark all the components that are part of the circuit defect-free.

2. Repeat this for many tilings, so that each component is part of many different test circuits.

The number of tilings needed will depend on the desired *yield*. By yield, we mean what fraction of the defect-free components are marked defect-free at the end of the above procedure. Note that we identify all the defective components as being defective (i.e., there are no *false positives*). However, in the process, some good components may also be marked bad and thus lower the yield.

Consider a defect-free component, X; X is marked defect-free if it is part of a test circuit that has no other defects. The probability of X being defective is p, the size of each test circuits is k, and t different tilings are used (i.e., X is made a part of t different test circuits which only have X in common). Let $P(A)$ be the probability of event A; then,

$$\begin{aligned}
&P(X \text{ is marked defect free}) \\
&\quad = P(\text{one out of the } t \text{ test circuits is defect free}) \\
&\quad = 1 - P(\text{all } t \text{ test circuits have at least one defect each}) \\
&\quad = 1 - P(1 \text{ test circuit has at least one defect})^t \\
&\quad = 1 - \{1 - P(1 \text{ test circuit has no defects})\}^t \\
&\quad = 1 - \{1 - (1-p)^{k-1}\}^t
\end{aligned} \tag{1}$$

If we want the yield (i.e., the probability of each defect-free component to be marked defect-free by this process) to be y, we get:

[2] Note that the wires connecting computational elements on the fabric are also "components" with a probability of failure less than or equal to p

[3] The Teramac achieved defect tolerance for defect densities of this order using essentially the method described above

$$1 - \{1 - (1-p)^{k-1}\}^t > y$$
$$\Rightarrow t > \frac{\log(1-y)}{\log\{1-(1-p)^{k-1}\}} \quad (2)$$

Let us consider the case where $k = 10$ and $p = 0.01$ (and, therefore, $k * p = 0.1$). If we desire a yield of 99%, we get $t > 1.88$, i.e., $t = 2$. This implies that with only 2 tilings (e.g., horizontal and vertical) we should be able to identify at least 99% of the good components; the remaining good components, and all the bad ones, are marked defective and not used.

3.1.2 $k * p \approx 1$

Each test circuit now has the expectation of having 1 error. However, this does not mean there are no defect-free circuits; in fact, the probability of any given circuit of size k being completely defect free is $(1-p)^k \approx (1-\frac{1}{k})^k \approx \frac{1}{e} \approx 0.35$, i.e., a third of the circuits can be expected to be free of defects. In this situation, the method outlined for the previous case should work; however, the number of tilings required would be more. For example, if we let $k = 10$, $p = 0.1$ and the desired yield $y > 0.99$, we get $t = 10$. Under our assumption of arbitrary connectivity, finding this many tilings is easy; however, this may not be the case in realistic situations.

3.1.3 $k * p \gg 1$

This is a significantly more difficult problem than the above two, since each circuit is now expected to contain multiple errors, and obtaining a test circuit that is entirely defect free is going to be very unlikely. For example, for the representative numbers $k = 100$ and $p = 0.1$, the probability of a circuit having no defects at all is $(1-p)^k = 0.9^{100} = 2.66 * 10^{-5}$. In fact, for these values of k and p, Equation (2) tells us that $1.76 * 10^5$ tilings will be needed for a yield of 99%; with our definition of tiling, of course, this number is unattainable. Therefore, the strategy of putting a component into different circuits till one that is free of any errors is found is not going to work. There are three possible ways of getting around this problem:

1. Make k smaller, so that $k * p$ is closer to 1.
2. Put the tester in a tight feedback loop, so that it may use the results of the previous tests to decide what the next test-circuit configuration is going to be. The tester can then selectively test areas of the fabric that appear to be promising until it finds a sufficient number of defect-free circuits.
3. Use more powerful test circuits.

Reducing the value of k may not be possible because of the lack of fine-grained access to the fabric components. The second approach is likely to be extremely slow, since new configurations will be generated during the testing, and the time required to place and route them will add to the testing time. Therefore, using more powerful test circuits seems to be the most viable approach. For example, the circuit might compute a mathematical function whose output will deviate from the correct value if any of the circuit's components are defective; if the amount of this deviation deterministically depends on the number of defective components, then a comparison of the circuit's output with the correct result can tell us the number of defects present in the circuit, instead of just the yes/no answer about the presence/absence of errors that our earlier circuits were returning. Circuits designed for fault tolerance that compute in a sparse, error-correcting code-space (such as Reed Solomon codes [9]) are good examples. Circuits used for this purpose will have to satisfy the following two conditions:

1. A defect in a circuit component should translate in a deterministic manner to an error in the circuit's output (and, therefore, a knowledge of the error in the circuit's output should translate to a count of the defects in the circuit).
2. Fault-tolerant circuits can count upto a certain number of errors, and return an incorrect count if the number of faults exceeds their threshold. The testing circuit, however, should be able to say when the error count is above its threshold. This can be ensured by using two different circuits simultaneously, both of which fail in different ways when their threshold is exceeded. A difference in their defect counts will then imply more defects than the threshold.

Note that our proposal to use fault-tolerant circuits for finding the defects is not inconsistent with our rejection of them for performing general computation, since they are now being used as part of the manufacturing process where the disadvantages we listed above are not relevant.

Using these defect-counting circuits, we propose splitting the process of defect-mapping into two phases: a *probability-assignment phase* and a *defect location phase*. The probability-assignment phase attempts to separate the components in the fabric into two groups: those that are probably good and those that are probably bad. The former will have an expected defect density that is low enough so that in the defect-location phase, we can use methods proposed in Section 3.1.1 or 3.1.2 to pin-point the defects.

The first phase, that of probability-assignment, works as follows:

1. The components are arranged in test-circuits in a particular tiling (for example, vertically), and defect counts (or the fact that the number of defects were more than the test circuit's threshold) for all the test circuits are noted. This is done for a number of different tilings (for our simulations, the number of tilings = test circuit size = k)

2. Given these counts for all the circuits, we find the probability that any particular component is good. This is done as follows:

   Let A be the event that any particular component a is good.

   Let a_1, a_2, ..., a_k be the number of defective components in each test circuit that a is a part of.

   Let B be the event of obtaining these counts for all these test circuits.

   Therefore, we need to find $P(\frac{A}{B})$.

   Now, from Bayes' rule,

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

   Determining the numerator and denominator of the RHS is trivial, since a is the only component these circuits share. After simplification, we get

$$P(\frac{A}{B}) = \frac{1}{1 + \frac{(1-p)^{k-1}k^k}{p^{k-1}(k-a_1)(k-a_2)...(k-a_k)}}$$

   This equation is solved for each component to obtain its probability of being good [4].

3. The components with a low probability of being good are discarded, and this whole process is repeated. This is continued for a pre-determined number of times (say, N_1) or till a certain fraction of the components are discarded, whichever is earlier.

At the end of this process, we have divided the components into two groups: those with a high probability of being good, and those with a high probability of being bad. The latter are discarded, while in the former, the fraction of faulty components is expected to be down to such a level that methods described in Section 3.1.1 and 3.1.2 can be applied. Therefore, the second phase, that of defect-location, proceeds as follows:

1. The test circuits are run again on the reduced set of components; if a defect count of zero is obtained for any circuit, all its components are marked as good. This is done for all k tilings.

2. Some of the components that were removed in Phase 1 (for having a low probability of being good) are added back and the tests are repeated. This is done for some pre-determined number of times (say, N_2).

3. Finally, all the components marked as good are declared good, and the others are declared bad.

We have not been able to work out a mathematical model of how good this method is going to be, given different values of k, p and the defect-counting threshold of the test circuits; in particular, the values of N_1 and N_2 are determined heuristically. However, there are a number of points to note here:

1. The quality of the results will depend on the defect-counting threshold of the test circuit; the higher this threshold, the more information can be obtained and the better the results should be.

2. A number of good components are marked bad. This is because in Phase 1, the counts for all circuits are not available (because the threshold is less than the actual number of errors in many cases) and due to this incomplete information, many good components get discarded. This is the *waste* of the procedure.

3. Each of the 2 phases is being run in multiple steps (N_1 and N_2 respectively) and each of these steps discards some components which are not used in later steps; therefore, it will not be possible to completely pre-determine the test circuits *a-priori*, and some feedback dependent configuration on the part of the tester will be required, although this will be limited to routing the pre-determined test circuit configurations around the discarded components [5].

To test the effectiveness of this procedure, and to measure the impact of the defect-counting threshold on the output, we ran a number of simulations, the details of which are presented in Section 4.

[4] To maintain clarity, what happens when the number of faults is more than the test-circuit's threshold has not been described above; these numbers are discarded and the formulas are adjusted accordingly.

[5] N_1 and N_2 are set greater than 1 to maximize the yield that this process provides us; we have found in our simulations that for more powerful test circuits or smaller defect counts, setting $N_1 = N_2 = 1$ provides acceptable results.

3.2 Scaling with Fabric Size

We next have to ensure that the testing procedure scales with the fabric size. A short testing time is crucial to maintain the low cost and usefulness of these fabrics. We shall begin by analyzing the testing strategy above to see how long it takes to run.

For a $k*k$ piece of the fabric, we make at most $N_1 + N_2$ sets (N_1 in the first phase and N_2 in the second) of k tilings each in the fabric, and each tiling consists of k different test circuits. However, in any particular tiling, no two test circuits have any component in common; therefore, all the test circuits for a particular tiling can be run in parallel. This leaves us with $(N_1 + N_2) * k$ steps, each requiring a reconfiguration of the entire fabric being tested. If the fabric is larger then $k*k$ it can be split into many sections of size $k*k$, each of which can be tested separately. As mentioned above, we do not have a model to describe how N_1 and N_2 scale with k; however, our simulations show that even for very weak test circuits and fairly high defect densities, the yields plateau out beyond $N_1 = 6$ and $N_2 = 4$. We therefore believe that the number of reconfigurations required would be linear in k, i.e., proportional to the square root of the size of the fabric's part under test.

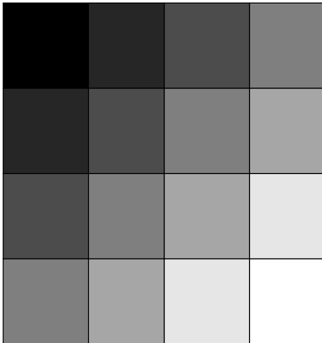

Figure 2: *A schematic representation of how testing will proceed in a wave-like manner through the fabric. The black area is tested and configured as a tester by the external tester; each darker-shaded area then tests and configures a lighter-shaded neighbor. For large fabrics, multiple such waves may grow out from different externally-tested areas.*

The above description shows that testing a particular part of the fabric takes time proportional to the square root of the part's size; however, since the number of such parts grows linearly with the size of the fabric, testing time would still appear to grow linearly with fabric size. This misses the crucial point that the fabric itself is reconfigurable - in this situation, reconfigurability helps in two ways:

1. Once a part of the fabric is tested and defect-mapped, it can be configured to act as a tester for the other parts, thus bringing down the time required on the external tester drastically.
2. Once the tester is configured onto the fabric, 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 2). 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 side, and to the square root of the components in the fabric.

Thus, by leveraging the reconfigurability of the chip, the time on the external tester, and also the total testing time, can be reduced by a significant amount.

4 Simulation and Results

We have checked the usefulness of each testing strategy we present by simulating it on a number of test cases. We do this in the following way:

1. The fabric is represented by a matrix of 0s (for good components) and 1s (for defective components).
2. The errors (i.e., the 1s in the matrix) are generated randomly with probability p.
3. A "test circuit" is simulated simply by examining its constituent components and returning the relevant information; for example, if a circuit is supposed to tell whether there is an error in a particular column or not, we return "no error" if all the numbers in that column are zeroes. The only way to access components of the matrix is through this test circuit interface.
4. Using the information our test-circuits give us, we try to reconstruct the error-map for the fabric. Our final aim is to ensure that the output is *correct*, i.e., that no defective component is declared good, and that there is *high yield*, i.e., that as many good components are declared good as possible.

All the results we present are in terms of the % yield, which is defined as follows:

$$\text{yield} = \frac{\text{No. of good components identified}}{\text{No. of good components actually present}} * 100$$

We first consider the cases where $k * p \ll 1$ and $k * p \approx 1$. The two candidate sets of values used for our simulations are $(k = 11, p = 0.009)$ and $(k = 11, p = 0.09)$ [6]. As described in Sections 3.1.1 and 3.1.2, we use test circuits that can detect the presence or absence of any errors, and measure the yields for different number of tilings t. This entire process is run 1000 times for each set of values of k, p and t, and the yields and averaged out.

	Number of tilings t	Expected Yield (%)	Achieved Yield (%)
$k = 11$ $p = 0.009$	1	91.36	91.34
	2	99.25	99.29
$k = 11$ $p = 0.09$	1	38.94	38.05
	2	62.72	62.05
	5	91.51	91.17
	10	99.28	99.24

Table 1: *Simulation results comparing expected and achieved yields for 2 sets of values of k and p*

The results of our simulations are presented in Table 1. We present both the expected yield, obtained by plugging the values of k, p and t into Equation (1), and the yield achieved in our simulations. As can be seen, the two figures closely match.

We next simulated the algorithm we describe in Section 3.1.3 for the case where $k * p \gg 1$. For the simulation, we used a fabric of size 101*101, k and t (the number of tilings used) were set equal to 101, N_1 (the maximum number of times this set of tilings would be used in the probability-assignment phase) was set to 6, and N_2 (the maximum number of times the set of tilings would be used in the defect-location phase) was set to 4. We then measured the yield obtained as the defect densities, and the number of defects our test-circuits could count, varied. There results are summarized in Figure 3.

From our results, it is apparent that it is possible to achieve high yields even with test-circuits that can count a small number of defects, particularly if the defect density is low. For example, for densities less than 10%, a test-circuit that could count up to 4 errors achieved yields of over 80%. With more powerful test circuits, yields of over 95% are achievable. Also, although N_1 and N_2 have been set to 6 and 4, respectively, each set of tilings were run this many times only for weak circuits (those that could count 3 defects or less) or for very high defect densities (12 % and over). For moderately powerful test circuits and defect densities around the 10% mark, the set of tilings were used just once during each phase of the algorithm. Therefore, we can say that the number of reconfigurations required is linear in k, or proportional to the square root of the fabric size.

[6] We choose the value 11 because we need k to be prime to keep our analysis simple: it makes it easy to find a large number of tilings for the test-circuits.

5 Open Issues

Design of the test circuit: Test circuits still need to be designed that give us the kind of information we require. If the large amount of literature on fault-correcting circuit design is anything to go by, this in itself is a fairly difficult task.

Alternative circuit types: It may not be possible to design circuits that return actual defect counts; however, it might be possible to use other, less powerful circuits that are still capable of providing useful information.

Accounting for limited fabric connectivity: The description and analysis presented above assume that we have unlimited connectivity on the fabric. The actual scenario is likely to be very different; we might be able to connect a component only to its neighbors. This will require reducing the number of tilings we need to make, and also changing the mathematical analysis so that the tilings are less restricted (e.g., circuits from different tilings may have more than one component in common).

Accounting for real defect types and distributions: These will not be known until we begin manufacturing large-enough nanofabrics in some numbers. However, the availability of this information will have interesting ramifications: for one, designing test-circuits for certain types of defects, such as *stuck-at* defects, is fairly well-studied and well-understood. Also, it might be found that defects tend to occur in clusters; this will make it easier to detect the defects, and less likely for good components to be marked bad.

6 Conclusions

CAEN-based computing devices are expected to be inexpensive to produce, yielding devices 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 devices 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.

There are two main axes to achieving scalable defect toler-

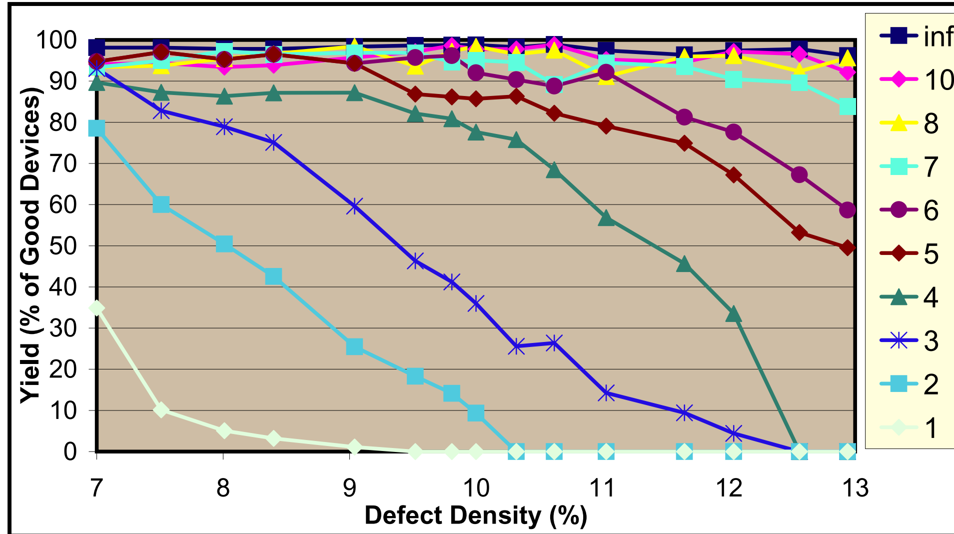

Figure 3: *Yields achieved by varying the defect densities and the number of defects our test-circuits could count. The test-circuits consisted of 101 components, and for each simulation run, a set of 101 tilings was used a maximum of 6 and 4 times respectively in the two phases.*

ance. First, the defect testing must scale with defect density. Second, the defect testing must scale with device size. Our method, inspired by the Teramac, configures the fabric under test with circuits that when run will indicate not only whether the tested area is defective, but indicate how many defects it contains. By using test circuits that can "count" we show how defect testing can scale with defect density. Our method requires that the underlying fabric have a rich interconnect. By harnessing the reconfigurability of the fabric we also show how the process can scale with fabric size. Once a small portion of the fabric has been diagnosed, it can be configured to test itself. If the device can configure itself, then the testing can proceed in parallel and the total time to test the device scales with the square root of the device size.

Acknowledgments

This work was sponsored in part by DARPA, under the Moletronics Program, and Hewlett-Packard Corporation. The authors want to thank Phil Kuekes and Avrim Blum for their helpful comments.

References

[1] J. H. Schön, H. Meng, and Z. Bao, "Self-assembled monolayer organic field-affect transistors," *Nature*, vol. 413, pp. 713–716, Oct. 2001.

[2] 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.

[3] 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)*, pp. 178–191, July 2001.

[4] 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)*, 1997.

[5] 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.

[6] 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), pp. 185–192, Apr. 2000.

[7] N. Pippenger, "Developments in "The Synthesis of Reliable Organisms from Unreliable Components"," *Proceedings of Symposia in Pure Mathematics*, vol. 50, pp. 311–324, 1990.

[8] D. A. Spielman, "Highly Fault-Tolerant Parallel Computation," in *Proceedings of the 37th Annual IEEE Conference on Foundations of Computer Science*, pp. 154–163, 1996.

[9] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics (J. SIAM)*, vol. 8, no. 2, pp. 300–304, 1960.