Online Test and Fault-Tolerance For Nanoelectronic Programmable Logic Arrays

Saturnino Garcia and Alex Orailoğlu
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92037
{sat, alex}@cs.ucsd.edu

***Abstract*— Fault-tolerance is one of the major challenges facing nanoelectronic systems. Previous defect-tolerance techniques have focused on offline testing and are ill-suited for handling device death. We propose a fault-tolerance scheme for nanoelectronic PLAs that is based on checkpointing. With low overhead, our scheme is able to test and diagnose crosspoint faults that may appear in the PLA during its operational lifetime. We also present a new test vector compaction algorithm that significantly reduces the number of test vectors. This new algorithm takes advantage of the density and reconfigurability of nanoelectronic circuits by raising the granularity of defect diagnosis to the row/column level. We are therefore able to reduce the number of test vectors to O(n+p) for a PLA with n input and p product lines without sacrificing diagnosability. Experimental results show that our checkpointing scheme, together with the compaction algorithm, tolerates a much higher fault rate than previously possible.**

Index Terms — **PLA, testing, fault-tolerance, checkpointing, nanoelectronics**

I. Introduction

As CMOS continues to scale down towards its physical limits, nanoelectronic devices have continued to make progress as a possible alternative or complement technology. Researchers have already succeeded in making diodes and two-terminal transistors out of crossed sets of carbon nanotubes (CNT) and silicon nanowires (SNW) [1], [2]. Systems built on these crossbar-based nanoelectronic circuits have already been proposed [3], [4], [5], [6], [7], [8], [9], [10]. Nanoelectronic-based circuits offer the potential for device densities that are orders of magnitude higher than CMOS.

However, unlike traditional CMOS circuits, which have been fabricated with precise top-down lithographic methods, nanoelectronic circuits are expected to rely upon a bottom-up self-assembly. This new approach to fabrication will lead to higher rates of defects due to its imprecise nature. When defect levels are low enough, as has traditionally been the case with CMOS, simply discarding defective chips will still lead to acceptable yield. With defect rates as high as 10%, as has been predicted for nanoelectronic devices, a new strategy is needed. Luckily, the low-overhead reconfigurability that is intrinsic to many electronic devices provides a new way to approach fault-tolerance. By adding diagnosis to our testing procedure we may uncover which devices are defective and reconfigure our circuit to avoid these defects.

Previous approaches to defect-tolerance in nanoelectronic circuits have mainly dealt with defects that are present at manufacturing time [11], [12], [13], [14], [15], [16]. While testing and diagnosing these defects are an important part of circuit configuration, it does not deal with the problem of device death.[1] Dealing with device death presents new challenges for defect-tolerance. In the online testing environment, we must not interfere with the circuit's normal operation. Previous approaches relied on frequent, intrusive reconfiguration, which makes them unsuitable for online use. Furthermore, in order to avoid data corruption, the speed of test and diagnosis is of heightened importance. The high-fault rate of the nanoelectronic environment would severely limit online approaches like that in [17] because of the long test times involved. Finally, if we are to maintain the advantage in device density that nanoelectronics affords us, hardware overhead is also a major concern. The work in [18] addresses online test in nanoPLAs and is able to maintain uptime and have quick fault diagnosis. However, this approach requires a large overhead and makes special assumptions about the observability of different parts of the PLA. The work presented effectively deals with all three challenges of the online environment: uptime, diagnosis speed, and hardware overhead.

In this work we develop a fault-tolerance scheme for nanoelectronic PLAs (nanoPLAs) that is based on checkpointing. Compared to traditional hardware-redundancy schemes, it achieves arbitrarily low overhead by partitioning the PLA blocks into test groups and designating two blocks to act as surrogates. Testing happens concurrently with normal operation of the system, allowing for the maximum amount of uptime for the system. Upon detection of a defect, rollback allows us to return to a previously saved state and avoid data corruption. In order to minimize the rollback distance, we introduce a test generation algorithm that provides a compact set of test vectors that detects 100% of crosspoint faults. Our algorithm leverages the abundant resources of the nanoelectronic environment and decreases the diagnostic accuracy—from crosspoints to rows/columns—in order to achieve a more compact set of test vectors. Compared with the PLA diagnosis approaches in [19], [20], our test generation and compaction algorithm has much faster speed ($O(n+p)$ compared with $O(n+m)p)$) while steering clear of

[1] Device death is the permanent disablement of a device during its operation. Defect maps formed during pre-operational testing will not include these types of defects.

978-1-4244-2553-2/08/$20.00 ©2008 IEEE

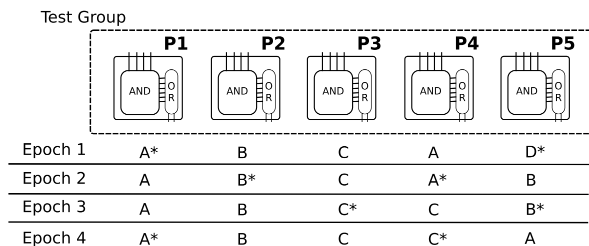


Fig. 1. Test group with five PLA blocks. A * indicates a block is undergoing testing.

any unrealistic assumptions about defect-free lines or access mechanisms to the PLA.

The layout of this paper is as follows. First, in Section II we briefly describe the nanoPLA that we have targeted. We start the discussion of our proposed solution in Section III when we describe our checkpoint scheme. We follow this with a description of our method of test vector generation. We present experimental results in Section V, showing the effectiveness of our test vector generation in minimizing the number of required test vectors. We also show the effectiveness of the overall checkpointing scheme in tolerating device death. Finally, we conclude in Section VI.

II. Nanoelectronic PLA Overview

In this section we will give a brief overview of the nanoPLA environment which we are targeting. We assume that the system consists of a number of logic cells which we refer to as PLA blocks. Furthermore, each of these PLA blocks consists of a nanoelectronic crossbar structure with n input, p product, and m output lines (we will use the notation (n, p, m) to describe the dimensions). Logically these PLA blocks may appear to be of smaller size as we will designate several lines as spares to be used in repairing defects. As with the crossbar structures described in [1] and [2], the crosspoints of these nanowires are electronically configurable so that it is possible to reconfigure them easily by applying the correct voltage bias. Sequential circuit elements may be implemented in nanoelectronic crossbar memories that use replication or error correcting codes for fault-tolerance.

We assume that the system has a rich interconnect structure consisting of both CMOS and nanowires, such as the system developed in [16]. An interface similar to the one developed in [21] allows us to control and observe the inputs and outputs of the PLA blocks by utilizing micro-to-nanoscale decoders. Using either fault-tolerant nanoscale memory or the CMOS/nano interface, we will be able to apply a set of test vectors to the PLA blocks. Checkpointing and rollback, as described in III, will need to be coordinated by a system-wide controller implemented in the more reliable CMOS substrate.

III. Checkpoint Scheme

Checkpointing is a traditional technique used for handling errors [22]. Checkpoints are consistent states in the operation of a system. Upon detection of an error, we may rollback to the most recent checkpoint and continue execution. Because checkpoints represent safe points during our operation, we must ensure that all PLA blocks in the system have been thoroughly tested before generating a new checkpoint. However, because complete test and diagnose requires control over the inputs to the PLA block, normal operation of this block must be suspended. If we wish to maintain forward progress for the system, another PLA block must act as a surrogate during testing. This surrogate will implement the same functionality as the block under test (BUT). Because they represent overhead for testing, we would like to minimize the number of blocks that are concurrently under test. Unfortunately, as we decrease the number of BUTs we also increase the amount of time needed to test all of the blocks in the system. We are thus presented with a trade-off between speed and overhead. However, further complicating the matter is that fault rates may not be constant. At time t_1 the fault rate may be low enough to allow for a small number of BUTs while at another instant, t_2, fault rates may increase so that quicker detection is necessary. In order to handle this case, we have developed a checkpointing scheme that involves *test groups*.

A. Test Groups

Our proposed checkpoint scheme revolves around the idea of a test group. A test group is simply a grouping of N PLA blocks, two of which are designated as surrogates. Figure 1 shows a simple test group formed with five PLA blocks. Blocks P1 through P3 implement PLA functions denoted as A, B and C while P4 and P5 are the designated surrogates of the group. At any time during operation, one of the $(N-2)$ non-surrogate blocks (which we will refer to as P_t) will be in a testing state while the other non-surrogate blocks are operating normally. While P_t is being tested, one of the surrogates will be configured to implement P_t's normal functionality. This will allow complete functionality at all times. After P_t finishes its testing, one of the other non-surrogate blocks is selected and it then becomes the BUT. The block that will be the BUT is chosen in a round-robin manner so that all other non-surrogate blocks are tested before P_t undergoes testing again. We add a second surrogate to the test group so that the first surrogate may also be tested for defects.

Time is divided into multiple *testing epochs*. During a single testing epoch, one of the PLA blocks in the test group will undergo complete testing for all meaningful crosspoint faults. We classify a crosspoint fault as being meaningful if it will affect the output of the PLA. Conversely, we can safely ignore meaningless faults such as stuck-open faults on unused crosspoints or stuck-closed faults on used crosspoints. After the conclusion of a testing epoch, another block is put under test and another epoch begins. As stated earlier, before a block can be tested for a second time, all other blocks in its test group must also be tested. Figure 1 shows the progression of test epochs. In epoch 1, P1 undergoes testing while P4 acts as a surrogate by implementing configuration A. Epoch 2 sees P2 undergo testing while P5 implements its functionality and

P4 also tests itself.

While Figure 1 shows only one test group, in a system with many PLA blocks, the blocks will be divided into multiple test groups. It is not necessary that these groups remain the same indefinitely. As we alluded to earlier, differing fault rates may change the maximum test group size that is possible. In Section III-C we also discuss another way in which the flexibility to change test group size is beneficial.

B. Generating Checkpoints

At the end of each testing epoch, we gain some new information about the state of the system. If the test did not uncover any defects, then we can conclude that no erroneous data has been generated by defects since the last time this block was tested. However, we have no such assurances about other PLA blocks. It is possible that another defective block is producing incorrect data which is poisoning the rest of the system. Therefore, if we generate a checkpoint at the end of a testing epoch, it is only a *tentative checkpoint*. We cannot rollback to tentative checkpoints because they may involve incorrect data.

For a checkpoint to be considered a *safe checkpoint*, we need to be assured that the blocks that weren't being tested during this epoch were not defective. We must therefore wait until all other blocks have been tested before we can conclude that the checkpoint is safe. Our choice of a round-robin selection of BUTs provides us with an important property: only $(N-3)$ clean tests are needed before a tentative checkpoint can be considered safe. We therefore only need to save $(N-2)$ checkpoints at any time. The oldest checkpoint will be the safe checkpoint while the others will be tentative. When doing rollback, our rollback distance will always be limited to $(N-2)$ test epochs.

The length of a test epoch will be equal to the time required to completely test and diagnose the BUT for that epoch. While the expected mean time to failure (MTTF) will dictate how often rollback occurs, the size of the test epochs will ultimately determine what percentage of computations are wasted while recovering from a failure. Unlike traditional checkpointing systems where the distance between checkpoints is a matter of the extent of the performance overhead one is willing to accept, checkpointing is intimately tied to the testing epochs. In Section IV we will discuss a method for shortening the time required to completely test a PLA block.

C. Rollback Recovery

At the end of a testing epoch, if a defect is detected, then the rollback-recovery mechanism is initialized. First, the defective PLA block(s) must be repaired to ensure future error-free operation. Because the output of our diagnosis (as described in Section IV) indicates which rows/columns are defective, repair is simply a matter of reconfiguring a spare row/column in the PLA to act as the defective row/column. After this reconfiguration is complete, the system rolls back to the most recent safe checkpoint.

In order to roll back the system to an earlier time, our checkpoints must accurately capture the state of the system. Because a PLA is a strictly combinational circuit, it holds no internal state. Therefore, to restart a PLA at a specific instant in time (e.g. at the end of epoch 1), we only need to know the inputs to the PLA at that time. However, the inputs to the PLA may come from two different sources: primary inputs or the outputs of other PLA blocks. In the former case, we do not need to save any additional data since we have access to the primary inputs. In the latter case, we must save the output(s) of the other PLA block(s). When generating a checkpoint, we will therefore only need to save a subset of the inputs to a block, namely those emanating from other blocks.

Since a checkpoint consists only of the output of PLA blocks, these outputs could be routed to a fault-tolerant nano-electronic memory or the more reliable CMOS substrate. In this case, rollback recovery will require that routing resources be configured so that the information in the checkpoints is guided to the correct destination. Alternatively, we can simplify recovery by adding several rows to the output of each PLA block. When a checkpoint occurs, the voltage on one of these rows will be set so that the values on the output wires are programmed into that row. When rollback recovery is initiated, the row that contains the safe checkpoint will be read so that all intermediate results are immediately passed to the correct destination. Conversely, to avoid the overhead of adding extra rows to the output, spare rows from the AND plane can be programmed and read in a similar way. However, to insure the integrity of checkpoints stored in the unreliable nano substrate, additional fault tolerance would need to be employed (such as comparing this checkpoint with one stored in the reliable nanomemory or CMOS).

While we expect there to be ample spare resources available to help recover from faults, after a long operational lifetime it may be that a PLA block runs out of extra rows and columns. In this case, rollback becomes complicated since we must do more than simply restore the previous checkpoint. Luckily, our checkpointing scheme can use several options to continue operating normally in the face of dwindling spare resources.

One option is to reconfigure the block with the same functionality but a different mapping of rows and columns. This option would work well if the PLA is sparsely populated. Unfortunately, a large number of defects or a high degree of crosspoint utilization may make this option unattractive. A similar option would be to try to reconfigure the PLA into its original state. Because a crosspoint toggle caused by a transient fault has the same effect as a permanent fault, our scheme might incorrectly classify a row/column as being defective. By reconfiguring into the original PLA state, we may find that a fault was not permanent and therefore the row/ column does not need replaced.

A third option would be to repartition the PLA blocks into new test groups. The new partitioning would avoid the use of the PLA block that was just retired. In order to do this, we would need to have an extra PLA block available that was previously unused. If the circuit does fully not use all the PLA blocks available on the chip, finding an extra PLA block would simply be a matter of picking one from a set of spares. If there

are no spare PLA blocks—either from the circuit requiring all the blocks or from all the spares being previously used—we have one other course of action: increasing the test group size. When we increase the test group size, the percentage of blocks being used as surrogates decreases thus freeing up some blocks. The downside of increasing the test group size is that we increase the rollback distance and consequently the overall performance of the system.

IV. Test Generation

As we have just mentioned, the length of a testing epoch and therefore the rollback distance, is a function of the time required to test and diagnose a PLA block. In this section we will discuss a new method of generating test vectors. By raising the granularity of defect diagnosis, we can better suit the nanoPLA environment and generate a compact set of test vectors that will significantly reduce the number of tests vectors needed.

A. Targeted Defects

While PLAs may be subject to different types of faults such as stuck-at faults and broken wires, we narrow our efforts to detecting crosspoint faults. This follows from the assumption that crosspoint defects are the only types of defects that will appear during circuit operation. Broken nanowires, another type of fault expected in nanoelectronics, are caused by defects in the manufacturing process and should not occur during the operational lifetime of the circuit unless it undergoes physical stress. The other major class of faults, stuck-at faults, can be modelled as crosspoint faults.

Crosspoint faults are classified according to their location and nature, giving rise to the following four types of faults:

- G A missing crosspoint connection in the AND plane. Presents itself as an erroneous 1 output on some inputs (i.e. fault signature of $0 \rightarrow 1$).
- S An extra crosspoint connection in the AND plane. Fault signature of $1 \rightarrow 0$.
- A An extra crosspoint connection in the OR plane. Fault signature of $0 \rightarrow 1$.
- D A missing crosspoint connection in the OR plane. Fault signature of $1 \rightarrow 0$.

B. Granularity of Diagnosis

As we have mentioned, in the high-defect world of nanoelectronics diagnosis is a key element in allowing us to leverage the reconfigurability of our devices. When doing diagnosis we have an option as to the level of granularity at which we locate defects. At the lowest level of granularity, we identify specific crosspoints in our PLA which are defective. With information about a specific crosspoint that is faulty, we have the option of replacing either the row or column that the crosspoint is on. This is the approach taken in [23]. In a PLA with size (n, p, m) we have on the order of $(n + m) * p$ crosspoints to differentiate between. At the next level, we can identify a specific row or column that contains a fault but not necessarily which crosspoint is defective. In the same (n, p, m) PLA, we have only $n + m + p$ rows and columns to examine. At the highest level of granularity diagnosis becomes unnecessary because the whole PLA block will be replaced and no differentiation is needed.

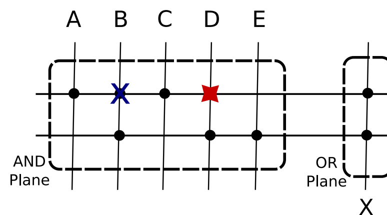


Fig. 2. Two faults: A *G* fault on the B input and a *S* fault on the D input.

At the heart of the choice between level of granularities is the trade-off between hardware cost and speed of detection. The lower levels of granularity sacrifice the latter for the former while the higher levels make the opposite trade-off. In the traditional scenario of offline testing with CMOS technology it makes sense to choose a lower level because hardware is not as abundant and time is not of primary importance. In the case of nanoelectronics and especially the case of online testing, we need to reevaluate the level of granularity we choose. In this new environment where hardware resources are cheap and plentiful, the necessity of fine-grained fault resolution wanes.

When dealing with nanoPLAs, the trade-off favors diagnosis at the row/column granularity. What is not so clear is how to efficiently diagnose errors to this level of granularity. In the following subsections we will discuss our method for row/column diagnosis. With our method, we can approach the theoretical lower bound on the number of test vectors needed.

C. Row-based Diagnostic Resolution

Let us first examine *S* faults. For example, consider the *S* fault on the D input line in Figure 2 which changes the function to $f_{s-fault} = ABCD + BDE$. In order to produce a test for this fault, we would need to set the product line to 1 while all other product lines are set to 0. Luckily, under the reasonable assumption that product lines in the PLA are unique, setting all inputs with connections to this product line to 1 while setting all other inputs to 0 would satisfy the requirements for our test. Therefore, the test vector ABCD'E' would test for this fault. Upon further investigation, it is easy to see that this test vector will also test for all *S* faults on that row. While this single test vector could not distinguish exactly which crosspoint on the row had the *S* fault, the repair strategy for all of them is the same: replace the row they are in.

Given that there are p product lines in the PLA, then exactly p test vectors will be needed to diagnose all the *S* faults. Because *D* faults have the same fault behavior as *S* faults, the same p test vectors for *S* faults will also test all *D* faults. *A* faults will also be covered by these test vectors as an extra connection in the OR plane will cause one of the outputs to go from 0 to 1. Unfortunately, because *G* faults are essentially the dual of *S* and *D* faults, the same test vectors will not yield tests

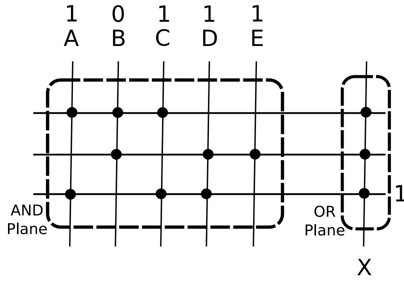


Fig. 3. Incompatible compaction of G faults on B input.

for the duals at the same time. As we will see, the row-based diagnostic approach will not be fruitful for *G* faults.

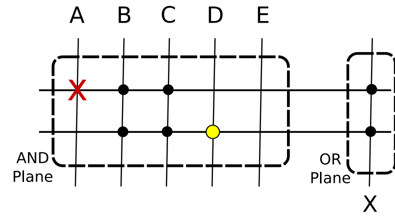
D. Column-based Diagnostic Resolution

For *S* faults, the fault syndrome is a 1 output going to a 0 ($1 \rightarrow 0$). For *G* faults, the fault syndrome is the dual of the *S* fault's syndrome (i.e. $0 \rightarrow 1$). To examine the differences between these two types of faults, Figure 2 also shows an example of a *G* fault. The G fault will result in the function $f_{g-fault} = AC + BDE$. To sensitize this fault, we set the inputs so that the normal output will be 0 while the faulty output will be a 1. Setting B to 0 while A and C are 1, will achieve this goal. Because B is set to 0, the other product term (BDE) will be 0 in both the normal and faulty cases so D and E are *don't cares* for our test vector. The test vector for this fault would therefore be 101XX. Using this reasoning, if we wanted to test for a *G* fault at A in the first product term, we could use either 0110X or 011X0 as our test vector.

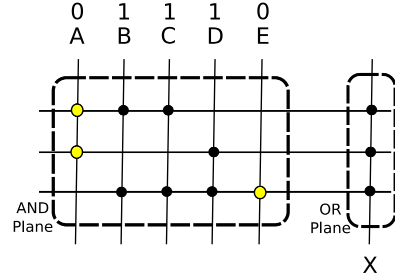
Comparing the test vectors for these two faults, we see that it is impossible to combine them into a single test vector because of disagreements between the A and B inputs. We can generalize this to say that any test vector will be best for at most one *G* fault on any given product line. This eliminates the possibility of doing row-based diagnostics on *G* faults. Fortunately, a column-based approach will lend itself more readily to *G* faults, as we shall see.

Let us consider Figure 2 once more, this time focusing on a *G* fault at the intersection of the B input and the second product line (not shown). To sensitize this fault, we would again set B to be 0. This time, however, D and E must be set to 1 while A and C are *don't cares*. The resulting test vector is therefore X0X11. In this case, we can combine this test vector and the test vector for the other *G* fault on B (101XX) to obtain a single test vector that tests for both faults: 10111. As in the row-based approach with *S* faults, while we cannot differentiate between the two *G* faults on the B input line, replacing that column will help us avoid both faults.

Unfortunately, it is not always possible to test for all *G* faults in a given column with a single test vector. Figure 3 is an example where this is the case. The example in this figure is similar to that in Figure 2 but with a new product line, ACD, connected to the output. Our previous test vector (10111) for the B input column will not work in this case because the



(a) Inherent Ambiguity: Cannot differentiate G faults on A and D.



(b) Compaction Ambiguity: Compaction of G faults on A leads to ambiguity on E.

Fig. 4. Comparison of two sources of diagnostic ambiguity.

ACD product term will be 1 on both normal and faulty outputs, masking all *G* faults. In this case, there is no single test vector that will test for all *G* faults on the B column. What we desire is an algorithm for finding the minimum number of test vectors required to cover all faults on a given column.

E. Column-based Vector Compaction Problem

Before devising an algorithm to generate column-based tests, we must formally define the problem we hope to solve. First, we will define several items:

- I — The set of all inputs to the PLA.
- J — The set of PLA inputs with a value of 1.
- q — The input corresponding to the column we wish to create a test for.
- C — The set of all product terms that contain q.
- P — The set of all product terms that do not contain q.
- S — A set of product terms that contain q.
- R — A set of sets of product terms that contain q.

As previously discussed, in order to test for a *G* fault we must set all non-q inputs to 1 while q must be 0. We must also ensure that there is at least one literal set to 0 in all product terms that do not include q (i.e. P). Including more faults to be tested in a single vector, we must set more of the inputs to 1. We can continue to add faults to the test vector until it no longer holds that at least one of the literals in each $p \in P$ is 0. We may formally write these requirements as the following constraints:

$$q \notin J \tag{1}$$

$$(\forall p \in P)(\exists i \in I \setminus J)(i \in p) \tag{2}$$

$$(\forall s \in S)(\forall z \in s)(z = q \vee z \in J) \quad (3)$$

In order to gain complete fault coverage, each $c \in C$ must be included in at least one S. Full coverage of a column with test vectors therefore involves choosing a set of S's that each satisfy the above constraints. We refer to this set as R. To minimize the number of test vectors required for a column test, we must minimize $|R|$ subject to the following constraint:

$$(\forall c \in C)(\exists S \in R)(c \in S) \quad (4)$$

This problem is a variant of the Hitting Set problem, a well-known NP-complete problem [24], and is therefore a NP-hard optimization problem. In order to solve this problem, we have developed a heuristic algorithm, which we describe in the next section.

F. Closest Term Algorithm

As we continue to add product terms to increase the number of faults covered per test vector, we make it harder to satisfy all the constraints of the problem. This follows from the fact that each product vector that is added increases the number of inputs that must be set to 1. In order to minimize the impact of adding another test, we will try grouping together product terms in C that share the most literals (i.e. are the "closest" to each other). For example, if we are creating a test for column A and the product terms involving A are ABCD, ABDE, and ABF, we will try to group together ABCD and ABDE since they share 3 literals. If grouping ABCD and ABDE together violates our constraints, then ABCD will be placed in its own vector and we will move on to trying to group together ABDE and ABF.

For a PLA of size (n, p, m) then our algorithm will run in $O(pn^2)$ time. The runtime will be dominated by the time it takes to check the constraint given in equation 2. If we defined $c = |C|$ then we will need to check $p - c$ product terms for the aforementioned constraint for every element in C. Since each product term can have up to n literals, the time spent for each column is $O((p - c)cn)$. We expect c to be small in comparison with n and p so we can simplify this to $O(pn)$. Since we have n input lines to generate tests for, we have $O(pn^2)$ overall runtime.

G. Diagnostic Ambiguities

While each test vector generated for *S*, *D*, and *A* faults will always lead to the unambiguous resolution of a fault to a specific row, ambiguities may be present when testing for *G* faults. Ambiguities that arise while testing for these types of faults may be inherent to the structure of the PLA or they may be a result of the column-based test vector compaction. Figure 4 illustrates the difference between the two sources of ambiguity.

In Figure 4(a), we see an example of inherent ambiguity. Here we have a simple PLA with two product terms: ABC and BCD. A test for a *G* fault on the A term of the first product line will require that A be set to 0 while B and C are set to 1. Because B and C are set to 1, in order to avoid masking the fault we must set D to 0. This will give us a test vector of 0110. Following similar reasoning, if we wish to test for a *G* fault at the D on BCD, our test vector will also be 0110. Unfortunately there is no way to differentiate between these two faults so both the A and D lines would need to be replaced if an error was detected on this test vector. Luckily, we will only see this type of ambiguity in a very specific case. If we have two product terms, each involving x literals, that have exactly $x - 1$ literals in common, then there will be ambiguity in the two columns corresponding to the non-shared literals.

Figure 4(b) shows an example of ambiguity that results from test vector compaction. In order to test all *G* faults on the A column, we would need to use the test vector 01110. However, we notice that this test vector also detects a *G* fault on the E term of the BCDE product term. If we were to test for each fault on column A separately using the test vectors 01100 and 00110, then we could eliminate this ambiguity. We are thus offered a trade-off between the speed of testing and the level of ambiguity. The choice between these two trade-offs will be influenced by the MTTF and the number of spare columns available per PLA block.

H. Multiple Faults

Until this point, we have focused the discussion of our test generation algorithm on the single fault case. Although we expect the single defect case to be the typical case, it is still possible that multiple devices within a PLA will die in between tests of that block. Fortunately, our set of test vectors will cover multiple faults also. We will now describe why.

We can think of a multiple defect case as being a combination of single defects. For the multiple defect to go undetected, it would have to be masked in all the vectors that would have uncovered the single defects. We can easily show that this is not possible. First, consider *G* faults. A *G* fault will cause the output to go from a 0 to a 1. For another fault to mask this, it would have to inhibit the product term that the *G* fault is on from going to 1. An *S* fault would not produce this effect since it can only cause a product term to go from a 1 to a 0 when its input line is a 1, which it can never be for the test vector for the *G* fault. Another *G* fault could also not produce this masking since it has the wrong fault signature. A *D* fault could cause this effect but because a *D* fault is tested with a row-based test vector, it will be uncovered with another test vector and both faults will be fixed when the row they are in is replaced. Using similar reasoning, we can show that any other combination of faults will be detected by our scheme.

V. Experimental Results

Our proposal for checkpoint-based fault-tolerance was evaluated in two parts. First, we evaluated the effectiveness of our test generation scheme for a range of PLA sizes. We then tested the checkpointing method for several benchmark circuits with a varying fault rate. In the following subsections we will further describe the setup of these tests as well as provide the results of our tests.

name	avg (n,p,m)	max (n,p,m)	max vectors
alu4	(13,83,2)	(18,97,3)	324
apex2	(23,18,3)	(25,83,10)	282
apex4	(9,71,2)	(9,80,1)	355
bigkey	(20,88,4)	(22,100,6)	375
des	(22,38,7)	(23,100,5)	266
diffeq	(23,51,6)	(25,100,4)	555
dsip	(24,51,4)	(25,68,4)	204
elliptic	(22,35,8)	(25,100,6)	402
ex1010	(11,81,1)	(12,95,1)	467
ex5p	(7,25,9)	(8,34,10)	116
frisc	(23,42,6)	(25,100,2)	462
misex3	(24,83,2)	(15,99,3)	320
pdc	(16,91,3)	(19,100,2)	331
s38417	(19,30,8)	(19,50,9)	220
s38584	(22,31,10)	(28,96,10)	265
seq	(20,21,5)	(20,90,5)	244
spla	(13,69,8)	(6,98,6)	325
tseng	(23,50,8)	(25,98,4)	457

TABLE I
CIRCUIT MAPPING RESULTS

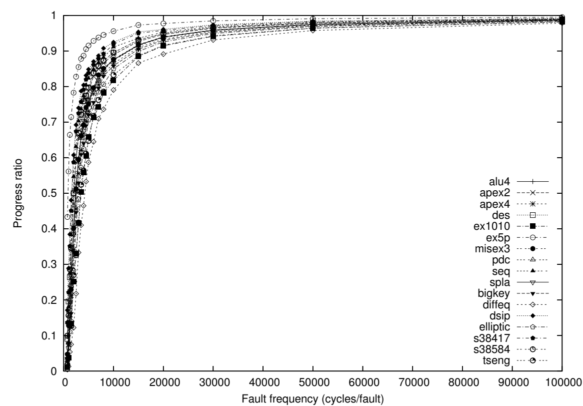


Fig. 6. Progress ratio for varying fault frequency. Group size = 6.

A. Test Generation

To evaluate the effectiveness of our test generation, we selected 18 of the largest benchmark circuits from the MCNC suite. For each of these benchmarks, we used the PLAmap utility [25] to map them into an array of PLA blocks. Using the PLAmap utility, we were able to specify the maximum dimensions of each PLA block. While the logic mapped into the PLA blocks will not exceed the maximum dimensions, it is possible that it will not use all the available lines. For the benchmarks, we tried to map the circuits into PLAs of dimension (n,p,m) = (25,100,10). These dimensions were chosen as a trade-off between the number of PLA blocks needed for mapping and the running time of PLAmap. Larger dimensions are also desirable because they allow the micro-to-nanoscale interface to provide higher device density.

Table I gives the maximum as well as the average dimensions of the mapped PLA blocks. The chart in Figure 5 compares the average number of test vectors needed for each of the benchmarks we evaluated to the theoretical minimum of our algorithm. The minimum which we can hope to achieve is $n + p$, where n is the number of input lines and p is the number of product lines. Here, we based the minimum on the average size of PLAs from Table I. From this figure we can see that our algorithm normally comes within a constant factor of 2 or 3 of the theoretical minimum, giving us $O(n+p)$. We can further compare this to previous PLA diagnosis algorithms which require $O((n+m)p)$.

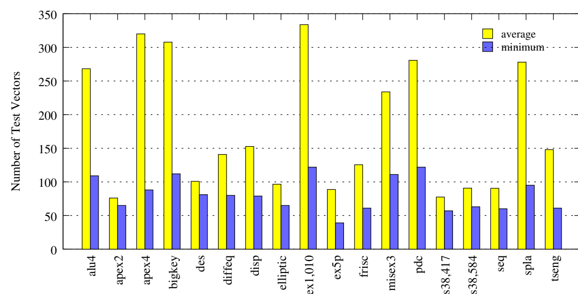


Fig. 5. Generated Vectors vs. Minimum Vectors

B. Checkpointing Evaluation

To test the effectiveness of our checkpointing algorithm, we again used the largest circuits from MCNC and mapped them into PLAs of dimension (25,100,10). We use a custom-built PLA simulator to test the circuits over any number of cycles. Our custom simulator allows us to randomly inject an arbitrary number of faults into any of the PLA blocks with any frequency.

To test the ability of our proposed solution to handle frequent faults, we simulated each of the benchmark circuits for increasing fault frequencies. To measure the amount of useful work done we define a quantity called the *progress ratio* to be the total actual number of cycles completed divided by the total number of cycles required to complete those cycles. For example, if we ran our simulator for 100 cycles but a rollback occurred from 75 to 50 then 125 total cycles would be required and the progress ratio would be $\frac{100}{125} = 0.8$. The lower this ratio, the more time is spent redoing the same work because of faults.

Figure 6 gives the progress ratios that result from varying fault frequency with 6 block test groups. Here we can see that the progress ratio quickly increases to 0.85 when the fault period is 10,000 cycles. For our simulation, we set the epoch length to be the time for the max number of tests (given in Table I), which averaged out to 331 cycles per epoch across all circuits. Because our rollback distance is 4 epochs, the average rollback time is $331 * 4 = 1324$ cycles. We could therefore expect 0.87 efficiency when the period is 10,000 cycles which is close to the 0.85 progress ratio from our results. If we had chosen a crosspoint granularity test method, the epoch size would have been approximately 3500 cycles. Using our test generation algorithm, we are therefore able to withstand a fault frequency that is 10X higher than previously possible.

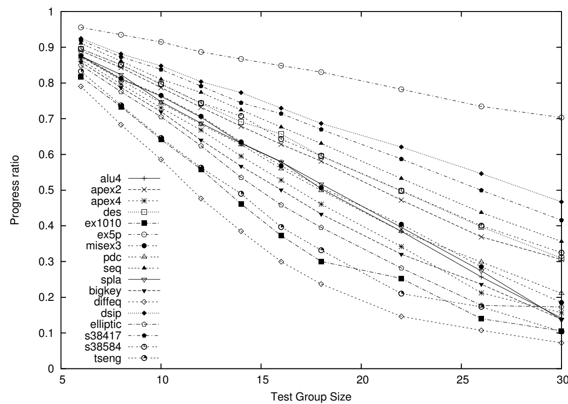


Fig. 7. Progress ratio for varying test group size with fault period of 10,000 cycles.

One of the parameters that we have control over for our proposed scheme is the number of test blocks per test group. Larger test groups help to amortize the required overhead for surrogate blocks but it also leads to longer rollback times. Figure 7 plots the progress ratio of each of the circuits for varying test group sizes when the fault frequency is 1 per 10,000 cycles. As we expected, increasing block sizes leads to a linear decrease in the progress ratio. At this high fault rate, for every test block added to a group we increase the progress ratio by roughly 0.03. Regardless of the fault frequency, we can see that decreasing group size can improve our progress rate.

VI. Conclusion

In this paper we have presented a checkpointing scheme designed to tolerate device death in a nanoPLA. By dividing the PLA blocks into test groups we were able to effectively share testing resources among multiple blocks and thus decrease the amount of hardware overhead needed. Depending on the prevailing MTTF, we are able to vary the number of test blocks per group and therefore dynamically alter the overhead to suit the fault rate.

We also developed a new test generation method that is based on test vector compaction among rows and columns. This row/column based compaction allows for diagnosis at the row/column level rather than at the crosspoint level. This level of granularity better suits the nanoelectronic environment, leveraging extra resources and reconfigurability in order to reduce test time. As a result, our new test generation algorithm reduced the number of test vectors from $O((n+m)*p)$ to $O(n+p)$, allowing for a 10X higher defect rate to be tolerated by our checkpointing scheme for PLAs of size (25,100,10).

References

[1] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K. Kim, and C. M. Lieber, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, no. 5545, pp. 1313–1317, 2001.

[2] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, no. 5545, pp. 1317–1320, 2001.

[3] S. C. Goldstein and M. Budiu, "Nanofabrics: spatial computing using molecular electronics," in *ISCA '01: Proceedings of the 28th Annual International Symposium on Computer Architecture*, 2001, pp. 178–191.

[4] J. P. Patwardhan, C. Dwyer, A. R. Lebeck, and D. J. Sorin, "NANA: A nano-scale active network architecture," *J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 1, pp. 1–30, 2006.

[5] J. P. Patwardhan, V. Johri, C. Dwyer, and A. R. Lebeck, "A defect tolerant self-organizing nanoscale SIMD architecture," in *ASPLOS-XII: Proceedings of the 12th international conference on Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 241–251.

[6] D. B. Strukov and K. K. Likharev, "CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, no. 6, pp. 888–900, 2005.

[7] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, no. 3, 2007.

[8] M. M. Ziegler and M. R. Stan, "CMOS/nano co-design for crossbar-based molecular electronic systems," *IEEE Transactions on Nanotechnology*, vol. 2, no. 4, pp. 217–230, Dec. 2003.

[9] R. M. P. Rad and M. Tehranipoor, "A new hybrid FPGA with nanoscale clusters and CMOS routing," in *DAC '06: Proceedings of the 43rd Annual Conference on Design Automation*, 2006, pp. 727–730.

[10] A. DeHon, "Array-based architecture for FET-based, nanoscale electronics," *IEEE Transactions on Nanotechnology*, vol. 2, no. 1, pp. 23–32, Mar. 2003.

[11] C. He, M. F. Jacome, and G. de Veciana, "A reconfiguration-based defect-tolerant design paradigm for nanotechnologies," *IEEE Design and Test of Computers*, vol. 22, no. 4, pp. 316–326, 2005.

[12] M. Mishra and S. C. Goldstein, "Defect tolerance at the end of the roadmap," *ITC '03: Proceedings of the International Test Conference*, pp. 1201–1210, 2003.

[13] J. G. Brown and R. D. Blanton, "CAEN-BIST: Testing the nanofabric," in *ITC '04: Proceedings of the International Test Conference*, 2004, pp. 462–471.

[14] M. Tehranipoor, "Defect tolerance for molecular electronics-based nanofabrics using built-in self-test procedure," *DFT*, pp. 305–313, 2005.

[15] Z. Wang and K. Chakrabarty, "Using built-in self-test and adaptive recovery for defect tolerance in molecular electronics-based nanofabrics," in *ITC '05: Proceedings of the International Test Conference*, Nov. 2005.

[16] R. M. P. Rad and M. Tehranipoor, "SCT: An approach for testing and configuring nanoscale devices," in *VTS '06: Proceedings of the 24th IEEE VLSI Test Symposium*, 2006, pp. 370–377.

[17] M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving stars: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems," in *EH '01: Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*, 2001, pp. 73–92.

[18] W. Rao, A. Orailoglu, and R. Karri, "Fault tolerant approaches to nanoelectronic programmable logic arrays," in *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007, pp. 216–224.

[19] V. K. Agarwal, "Multiple fault detection in programmable logic arrays," *IEEE Transactions on Computers*, vol. 29, no. 6, pp. 518–522, 1980.

[20] P. Bose and J. A. Abraham, "Test generation for programmable logic arrays," in *DAC '82: Proceedings of the 19th Annual Conference on Design Automation*, 1982, pp. 574–580.

[21] A. DeHon, P. Lincoln, and J. E. Savage, "Stochastic assembly of sublithographic nanoscale interfaces," *IEEE Transactions on Nanotechnology*, vol. 2, no. 3, pp. 165–174, Sep. 2003.

[22] N. S. Bowen and D. K. Pradhan, "Processor and memory-based checkpoint and rollback recovery," *Computer*, vol. 26, no. 2, pp. 22–31, 1993.

[23] S. Kuo and W. K. Fuchs, "Fault diagnosis and spare allocation for yield enhancement in large reconfigurable PLAs," *IEEE Transactions on Computers*, vol. 41, no. 2, pp. 221–226, 1992.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co. New York, NY, USA, 1979.

[25] D. Chen, J. Cong, M. D. Ercegovac, and Z. Huang, "Performance-driven mapping for CPLD architectures," in *FPGA '01: Proceedings of the 2001 ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays*, 2001, pp. 39–47.