Abstract—In this paper, a built-in self-test (BIST) procedure is proposed for testing and fault tolerance of molecular electronics-based nanofabrics. The nanofabrics are assumed to include up to $10^{12}$ devices/cm$^2$; this requires new test strategies that can efficiently test and diagnose the nanofabrics in a reasonable time. Our BIST procedure utilizes nanofabric components as small test groups containing test pattern generator and response analyzer. Small test groups (fine-grained test) result in higher diagnosability and recovery. The proposed technique applies the test in parallel with a low number of test configurations resulting in a manageable test time. Due to high defect density of nanofabrics, an efficient diagnosis procedure is done after BIST procedure to achieve high recovery. This is called recovery-increase procedure, and this increases the available number of fault-free components detected in a nanochip. Finally, a defect database called defect map is created to be used by compilers during the configuration of the nanofabrics to avoid defective components. This results in a reliable system constructed from unreliable components. Our simulation results demonstrate the effectiveness of the proposed BIST and recovery-increase procedures.

Index Terms—Built-in self-test, crossbar, molecular switch, nanowire, recovery.

I. INTRODUCTION

There are a set of theoretical, practical, and economical obstacles in further scaling the CMOS technology, and these have caused new paradigms in electronic devices, circuits, and design methods to emerge. One alternative to CMOS-based computing seems to be chemically assembled electronic nanotechnology (CAEN) [1]. CAEN uses direct self-assembly and self-alignment to construct electronic circuits from nanometer-scale devices. A CAEN-based device called nanofabric is an array of molecular electronic devices. It is estimated that nanofabrics can include up to $10^{12}$ devices/cm$^2$, which is significantly higher than that of the CMOS-based devices. However, defect densities as high as 10% are expected in these devices as well [2], [3]. This includes defects in all wires, logic blocks, and switches. Such high defect densities require completely new approaches to manufacturing and testing these new nanodevices.

Increasing defect density increases yield loss, and with such a high defect density in nanofabrics the cost of manufacturing test can be prohibitively high and discarding a faulty nanochip will no longer be possible. As a result, a nanofabric must be tested and diagnosed and the location of defects must be found. To create a reliable system, a method of using defective devices must be devised.

A natural solution to this problem is to design a reconfigurable device. The architectures can be originated from field-programmable gate arrays (FPGAs) [25] or the work on the Teramac custom computer [4], [5]. Prior research has shown that reconfigurable devices are fault tolerant in a sense that faults can be detected, and their location can be stored in defect map [4]. The defect map is a database that stores defect information (such as location, defect type, etc.) of the chip under test and is used during reconfiguration. The faulty blocks can be avoided during the reconfiguration of the device by referring to the defect map. The defect map is required to be constructed for each reconfigurable device. Testing and diagnosis of nanofabrics are required to achieve fault tolerance through reconfigurability. Similar to an FPGA, which is an interconnected set of configurable logic blocks (CLBs), nanofabrics also include programmable logic blocks and switches to implement combinational or sequential circuits.

In this paper, we propose a built-in self-test (BIST) procedure that tests the reconfigurable blocks and identifies the faulty ones. The proposed method creates a defect map, which can be used by programming the device for defect avoidance. A relatively low number of test configurations is required to test all the blocks in a nanofabric.

The proposed test method is fine-grained, i.e., test groups used in the BIST procedure are very small. Using fine granularity will result in higher diagnosis resolution and recovery. Recovery is defined as the percentage of blocks identified as fault-free after testing the nanofabric among all actual fault-free blocks in the nanofabric [19]. When a test group fails during test, all its blocks will be assumed faulty while there may exist fault-free components in that test group.

A. Nanofabric Architecture

The nanofabric architecture was first proposed in [1], and an island-style interconnect was assumed to provide the interconnection between clusters. Fig. 1(a) shows the island-style architecture as an array of size $M \times M$ of clusters [25]. In Fig. 1(b), each cluster is shown as a $N \times N$ array of nanoblocks and switchblocks. Nanoblock and switchblock are the two main components in a nanofabric. A nanoblock is similar to CLB in FPGAs. The nanoblock is based on a molecular logic array of size $K \times K$, as shown in Fig. 1(c) [1]. Molecular layer on the crosspoints of horizontal and vertical nanowires...
can be programmed as diodes. In this case, they provide the connection between the nanowires. The molecular layer can also be programmed as disconnection (open). These arrays can be created using chemical processes such as self-assembly and self-alignment of nanowires.

The region between a set of nanoblocks is known as a switchblock. Similar to a nanoblock, a switchblock is also reconfigurable and serves to connect wire segments of adjacent nanoblocks. The configuration switches of the switchblock determine the direction of data flow between the nanoblocks. Fig. 2 shows the internal structure of a switchblock.

Diode–resistor logic can be used to perform logical operations. Both the signals and their complements are created to produce a complete logic family. Fig. 3 shows implementation of simple logic gates (AND and OR). Logic values are restored using molecular latches [6]. The molecular latches provide a mechanism for latching the values (outputs of nanoblocks) and an isolation between the outputs of a nanoblock and inputs of adjacent nanoblocks.

To completely test a nanofabric, its clusters and interconnections between them must also be tested. Since the interconnect structure is very similar to island-style FPGAs, the test methods used in FPGA interconnect testing can be used for testing interconnects as well [28]. Hence, our approach focuses on testing the internal structure of clusters, i.e., nanoblocks and switchblocks.

Due to high defect rates, employing a fine-grained nanofabric architecture will improve test quality. A fine-grained nanofabric is defined to be a large array made of small clusters (\( M \) is large, but \( N \) is small in Fig. 1). Fine granularity provides high access to the small clusters of the fabric so that nanoblocks and switchblocks inside each cluster can be effectively configured as test groups and the test results can be read from the blocks. Designing fine-grained nanarchitectures will improve diagnosis capability and recovery through employing smaller test groups.

### B. Fault Model

A complete fault model in a nanofabric includes stuck-at faults, stuck-open faults, forward-biased diode faults (FBDFs), reverse-biased diode faults (RBDFs), and bridging (AND and OR) faults among vertical lines, horizontal lines, and vertical/horizontal lines. These faults are classified, based on different scenarios that occur due to flow-based assembly on nanowires in nanoblocks and switchblocks [3], [21], and are discussed in the following.

1. **Stuck-at Faults:** Any horizontal line or vertical line stuck at zero (Gnd) or one (Vdd).
2. **Stuck-open Faults:** A single line is broken, and it may result in a memory effect.
3. **FBDFs and RBDFs:** When a one is applied to one of the vertical lines and a zero is applied to a horizontal line, a forward-biased diode is formed between these two lines. A defective forward-biased diode will change the output line.
4. **Bridging Faults:** Any two lines shorted to each other create bridging faults of either types: AND-bridging and OR-bridging.

The proposed test procedure targets these faults and provides test architectures (TAs) and configurations for detecting them.

### C. Contribution and Paper Organization

Due to high-defect-density concerns and large number of components in nanofabrics, an efficient test methodology is required to achieve high-fault coverage and an effective diagnostic procedure is required to provide high recovery. In this paper, we propose a new BIST procedure to effectively test and diagnose the nanofabrics. The proposed technique is based on a fine-grained approach, i.e., it uses small test groups. The BIST procedure targets various fault types defined as the set of possible faults for nanoscale crossbars. New and efficient test groups, TAs, and test configurations are proposed to target these
faults. Each test group includes a test pattern generator (TPG), a response generator (RG), and the switchblock between PG and RG. Both PG and RG test themselves as they also generate test patterns and responses, respectively. PG tests itself and provides patterns for RG. The RG block receives the patterns to test itself and provides responses that can be read back by the tester for evaluation and recovery analysis.

Choosing small test groups improves test recovery. A fine-grained nanofabric architecture is capable of providing the required access mechanism for reading the test results from the small test groups of the cluster. Different approaches for providing such access mechanisms for nanoscale clusters are proposed in literature [26], [27]. The cluster blocks can be tested in parallel with low number of test configurations, which in turn reduces the overall test time.

Recovery-increase (RI) procedure is also proposed as a means to achieve higher recovery (i.e., a post-test processing step to identify more number of fault-free blocks in nanochip architectures). Our simulation results on few running examples demonstrate the effectiveness of our proposed BIST and RI procedures.

This paper is organized as follows: Section II describes the related prior work. Section III presents the proposed BIST procedure and the required TAs. Test configurations used for fault detection in nanoblocks and switchblocks are shown in Section IV. Recovery analysis and increase procedures are presented in Section V, and simulation method and results are illustrated in Section VI. Testing issues and basic comparisons with other techniques are briefly discussed in Section VII. This paper ends with concluding remarks in Section VIII.

II. PRIOR WORK

Integrated circuits are usually thrown away after the manufacturing test in case of detecting a fault in their components. Since the defect density of CMOS-based devices are still in a reasonable range, discarding a faulty chip is not very costly. However, there are some static and dynamic-fault-tolerance techniques that help a chip tolerate faults [7], [8]. Different approaches like triple-modular redundancy (TMR), cascaded TMR (CTMR), and sparing techniques are used in these fault-tolerance methods. Among various architectures, reconfigurable architectures like FPGAs can be very well adopted for fault-tolerance techniques. A test procedure can find faulty blocks and programming device can avoid them during reconfiguration.

Manufacturing test and application-dependent testing of FPGAs have been reported in [9]–[14]. Stroud et al. [9] and Abramovici et al. [10] proposed an FPGA BIST that tests CLBs and switches and diagnose the faulty ones. This approach divides the logic blocks of an FPGA into three blocks, i.e., TPG, block under test (BUT), and response analyzer (RA). The diagnostic capability can also be added to the methodology, by configuring FPGAs blocks first vertically and then horizontally; the faulty block located within the vertical and horizontal configurations can be detected. Application-dependent testing of FPGAs has also been proposed in [11] and [12]. Such techniques rely on the fact that a defective FPGA may pass the test for a specific application. This is a one-time configuration, in other words, the FPGA will no longer be reconfigurable. This reduces yield loss and yields to manufacturing-cost savings.

In recent years, many sparing techniques have been proposed for memories [15]–[17]. Extra rows and columns are fabricated for each memory; if a defective row or column is detected, the extra rows or columns can be used. Sparing techniques have been suggested for crossbars of nanowires [29] as well. However, sparing cannot be applied to a cluster in nanofabric because each cluster is composed of an array \((N \times N)\) of crossbars. High percentage of columns and rows (i.e., blocks) in a cluster are expected to be defective due to high defect probabilities. In other words, sparing can be applied at wire/switch level to crossbars (as in [29]), but it may not be suitable for rows and columns of nanoblocks and switchblocks.

A testing and defect-mapping strategy is used in Teramac custom computer [4], [5]. Teramac is an FPGA-based system with a high defect density of nearly 3%. Testing is performed by configuring the components as linear-feedback shift registers (LFSR). If the final bit stream generated by LFSR is correct, all the components are assumed to be defect-free. Otherwise, there is at least one faulty component in LFSR. To locate the faulty ones, the components in LFSR and other components are used to configure a new set of finer granularity LFSRs. If one of the new LFSRs is faulty, the component in the intersection of faulty LFSRs are identified as defective. A defect database is created after completing the test and diagnosis procedure.

Defect-tolerance methodologies are presented in [18] and [19], such that components of a nanofabric are configured to test circuits to infer the defect status of individual components. Since the fabrication technology of crossbars can be tuned to reduce the probability of stuck-closed faults, only stuck-at and stuck-open faults are targeted in these methods. In [19], test circuits are assumed to have the capability of counting the number of faults occurring in the BUT. LFSRs or counters are suggested for these test circuits. The capability of counting the number of faults is used to assign a probability of having none, some, or many faults in the BUT. This probability is used to
find fault-free nanoblocks in blocks with lower probability of having faulty nanoblocks. Also, very rich interconnections are assumed in [19] that can be used to implement these test circuits and connect BUT to external testers. Since the test circuits and BUT consist of large number of nanoblocks in this method, the required interconnects should provide global routing.

The proposed technique in [20] assumes that nanoblocks can be configured as test logic and, also, a memory in each nanoblock is used for testing purposes. The memory stores the status (faulty/fault-free) of adjacent blocks. Assumption of using memory cells in crossbars may face difficulties due to limitations in usage of NDR latches as free storage elements in each crossbar. However, the test procedure is relatively fast and achieved recovery is high.

Authors in [21] proposed a BIST procedure to test nanofabrics targeting more number of faults. A test group is considered to contain TPG, BUT, and output RA (ORA). Since the proposed test group contains three nanoblocks used as TPG, BUT, and ORA, two switchblocks should be used to provide the interconnection between the nanoblocks. There are various choices for selecting these switchblocks from the neighbors of the TPG, BUT, and ORA. Each of these selections will target and detect a set of faults in blocks. Since the technique uses AND and OR as ORA, it reads back only 1 bit, thereby the response readback will be fast, but the technique loses diagnosis information (i.e., in case of faulty response, it is not clear what line or switch was faulty).

In contrast, our proposed technique considers test groups containing only two nanoblocks, one configured as pattern generator (PG) and the other configured as RG. A single switchblock connects PG to RG. Hence, the number of different TAs required to cover all the faults in a test group is significantly reduced compared to [21]. On the other hand, reading out all the outputs of RG blocks identifies the exact location of faults.

III. BIST Procedure

In this section, we describe the BIST procedure to test nanofabrics and diagnose its defective components. We configure each nanoblock either as a TPG or RG. Due to the reconfigurability of nanofabrics, no extra BIST hardware is required to be permanently fabricated onchip. Moreover, a dedicated onchip nanoscale test hardware can be highly susceptible to being defective [22].

The test-configuration generation is first externally performed and then delivered to the fabric for each test session. The test session is referred to as one particular TA with configurations of nanoblocks and switchblocks. The architecture used in BIST procedure is called TA. Each TA includes test groups, and each test group contains one PG, one RG, and one switchblock associated with PG and RG. PG tests itself and sends a pattern through a switchblock to RG to test it, and then, the response is generated. Each cluster is divided into test groups. Consider an \( N \times N \) cluster including \( N^2/2 \) nanoblocks and \( N^2/2 \) switchblocks. In each TA, \( N^2/4 \) nanoblocks are configured as PG and the other \( N^2/4 \) nanoblocks are configured as RG. Overall, \( N^2/4 \) test groups are configured in each TA.

Generally, TAs are generated based on: 1) detection of faults in each nanoblock and switchblock and 2) direction of connections between the nanoblocks and switchblocks. Therefore, a number of TAs are generated for each test group and a number of test configurations are generated to cover all the faults in our fault model. During BIST, in each TA test groups are configured similarly, i.e., same faults are targeted within test groups in each test session. In each test group, three components are configured, i.e., PG, RG, and the associated switchblock. A switchblock in a test group is configured: 1) to transfer the patterns from PG to RG, while PG or RG are being tested, and 2) as a BUT when we target faults in the switchblock. The switchblock configuration depends on the used TA.

The programming device can configure the nanoblocks and switchblocks as required by the TAs and test configurations. Test results should be readback from the RG blocks using tester (on/off chip). Interconnect resources provided in the nanofabric architecture should be used for transferring these results to the tester. Since we are suggesting the use of a fine-grained architecture with small clusters, each cluster is composed of low number of nanoblocks. Therefore, the number of test groups in a cluster will be small and interconnect resources of the fabric is assumed to be sufficient to implement the readback mechanism. Hence, when a test session is done, the output of RGs is read for evaluation and recovery analysis by the tester. The programming device configures \( 3N^2/4 \) components in each TA and reads out the output of RG blocks; hence, \( N^2/4 \) components are accessed.

All nanoblocks of a cluster are configured as test groups using programming device, and then the test can be executed in parallel in all test groups. Therefore, the test-application time of a TA in a cluster is independent of the size of the cluster. However, parallel configuration of blocks of the cluster as test groups and parallel reading the test results of each TA depend on the interconnect and routing resources available for the cluster and the programming device’s bandwidth. The richer the interconnect, the higher parallelism can be obtained when testing clusters. Notice that when testing a nanofabric, in each step one, TA is used for the entire fabric. This provides us with the opportunity of parallel configuration of clusters, because configuration voltages will be the same for all clusters. The diagnostic and recovery analyses are performed after all test configurations are applied and all required data are collected.

The BIST procedure can be performed using an on-chip tester (microprocessor or dedicated BIST hardware) implemented in reliable CMOS scale on the substrate of nanofabric. This will reduce the test time, since external devices are generally slower than on-chip testers. The on-chip tester can execute BIST procedure, collect the test results, perform recovery analysis, and identify new test configurations to be used in the RI procedure. It may also eliminate the need to store defect map on-chip, since it can find the faulty blocks before configuration in each cluster [23], [30].

A. Test Architectures

Fig. 4 shows four TAs in a nanofabric, called TA11, TA12, TA21, and TA22, respectively. Note that the outputs in each TA
must be located on south or east of a nanoblock and the inputs must be located on north or west of a nanoblock as discussed earlier [see Fig. 1(c)]. To simplify the testing and diagnostic procedures, we assume that each PG generates output only on one side (either south or east), but an RG can receive the input data either on one side (either north or west) or on both sides (north and west). To configure a PG to generate the output, physical ones and zeros (Vdd and Gnd) must be used as inputs to nanoblocks. Depending on the type of target fault, RG may or may not connect to the physical Vdd and Gnd.

TA12 is the complement of TA11 since a PG in TA11 is used as RG in TA12 and vice versa. This guarantees that each block is configured as both PG and RG. TA21 and TA22 are also configured in a similar fashion except that TA22 is the complement of TA21. In TA11 and TA12, PGs generate the outputs on east side and RGs receive the patterns on their west side. In TA21 and TA22, PGs generate the output patterns on south and the outputs are transferred to the inputs of RG on north.

Fig. 5 shows an alternative set of TAs (TA31, TA32, TA41, and TA42). The difference between TAs, shown in Figs. 4 and 5, is that different set of inputs and outputs are used in each test group. For instance, outputs on south of PGs and inputs on west of RGs are used in TA31. Either one of these test groups can be used in each test procedure.

There are cases that two PGs are required for testing some faults in a nanoblock (see Section IV). Fig. 6 shows three more TAs (TA51–TA53) that use two PGs. Note that a combination of two PGs and one RG needs three TAs. Using all seven TAs (TA11, TA12, TA21, TA22, TA51, TA52, and TA53) or (TA31, TA32, TA41, TA42, TA51, TA52, and TA53) guarantees detection of all faults in nanoblocks. Additional TAs must be applied to achieve full coverage for switchblocks as well. Note that each TA (e.g., TA11) can be applied more than once depending on the type and number of faults considered to be tested in the fault model. Moreover, some faults may also need more than one test configurations. A portion of switchblock is also tested in each TA during the testing of nanoblocks. However, there will still be some undetected faults in switchblocks that must be tested separately using some additional configurations. This is further discussed in Section IV.

B. Test Procedure

There is no BUT in our BIST procedure as opposed to techniques proposed in [20] and [21]. Both PG and RG test themselves in each test session in a TA. Zeros and ones can be generated inside each nanoblock that results in testing that nanoblock, and the output is sent to another nanoblock, which can be reconfigured such that it tests itself. A combination of these two blocks working as PG and RG provides complete fault coverage for the defined set of faults. Since both PG and RG detect faults, our test procedure requires lower number of test configurations. A high-level description of the BIST procedure is shown as follows:

**BIST Procedure**

```plaintext
/* TA = {TA31, TA32, TA41, TA42, TA51, TA52, TA53} */
/* F = {f_j}, where f_j is the jth fault in the fault list */
for (i = 1; i ≤ 7, i++) do
    begin
        Select_TC(f_j) /* Selects appropriate test configuration (TC_j) for targeting fault f_j */
        Partition_Cluster(TA_i) /* Partitions the clusters into test groups based on TA_i */
        Config_TC(TC_j) /* Configures the test groups based on selected TC_j */
        Apply_Pattern /* PG applies the patterns and RG generates the responses */
        Read_Back(RG) /* Reads back responses generated by RGs */
    end
Identify_Faulty_TGs() /* Identifies the location of faulty test groups (FTG) */
RI_Procedure(FTG) /* Executes RI procedure */
```

TA contains all seven TAs used in **BIST Procedure**. F also consists of all the targeted faults listed in the fault model. The procedure starts with selecting a test configuration for targeting a fault from fault list \[ Select_TC(f_j) \]. It then partitions the clusters into test groups based on selected
TA \([\text{Config\_TG}(TA_i)]\). The test groups are then configured based on selected test configuration \([\text{Config\_TG}(TC_j)]\). Test configuration \((TC_j)\) represents the configuration required for detecting fault \(f_j\). Next, the patterns are applied and the responses are generated \((\text{Apply\_Pattern})\). The responses are read back using an on-chip or external tester \((\text{Read\_Back}(RG))\). The process continues until all the faults are targeted \((F = \{\})\), and the defect information is given to \(\text{Identify\_Faulty\_TGs}()\) function to find the location of FTGs and constitute the initial defect database to be used in RI procedure \((\text{RI\_Procedure}(FTG))\). RI procedure will be discussed in Section V.

**IV. TEST CONFIGURATIONS**

**A. Test Configurations for Detecting Faults in Nanoblocks**

In this section, we present the required test configurations for testing faults in a nanoblock. Stuck-at faults, stuck-open faults, forward-biased and reverse-biased diode faults (FBDFs and RBDFs), and AND and OR-bridging faults are targeted.

Each test group includes one PG and one RG. One may suggest that only one PG could be sufficient to test a nanoblock. This may test some of the faults in a nanoblock such as stuck-at faults but not all of them. Note that there are some faults in a nanoblock that need input(s) from other blocks to be detected. Few of the test configurations discussed here are similar to those proposed in [21], however there are several additional cases which are discussed in this paper.

One of the important features in each nanoblock is that the vertical lines can be connected to \(V_{dd}\) only and horizontal lines can be connected only to \(G_{nd}\), while both vertical and horizontal lines can be connected to one and zero generated at the output of surrounding nanoblocks. Thereby, PG is connected to \(V_{dd}\) or \(G_{nd}\) to generate the patterns and RG may or may not be connected to \(V_{dd}\) and \(G_{nd}\); in most cases, RGs receive output values of PGs.

Fig. 7(a) shows the test configuration for a nanoblock that tests stuck-at-zero faults on vertical lines [21]. All the vertical lines are connected to \(V_{dd}\), hence, the output lines are one. The output will be zero, if one of the vertical lines is stuck-at-zero or stuck-open. This architecture is used as PG, and because of that, the fault is called sa0-V-PG (stuck-at-zero on vertical lines of a nanoblock used as PG). If a fault exists, RG will receive the faulty output and detect the fault. As shown, PG tests itself because any stuck-at-zero and/or stuck-open fault on a vertical line can change its corresponding output-line value.

Stuck-at-zero and/or stuck-open faults on vertical and horizontal lines can also be detected by the configuration shown in Fig. 7(b). PG provides inputs either on the horizontal lines or on the vertical lines of RG (depending on the TA used). Hence, either horizontal or vertical lines of RG are tested. The test configuration shown in Fig. 7(b) is used as an RG, and the detected faults are called sa0-H-RG and sa0-V-RG.

Fig. 7(c) shows detection of stuck-at-zero and/or stuck-open faults on horizontal lines. Only one vertical line is connected to \(V_{dd}\), and all the horizontal lines are connected to \(G_{nd}\). A forward-biased diode is created on the junction of vertical lines
and horizontal lines. The horizontal output lines are one for a defect-free case. An output will be zero in the presence of a stuck-at-zero and/or stuck-open fault on horizontal lines. This test configuration is used as PG, and the fault detected is called sa0-H-PG. The faulty/fault-free output is transferred to RG and is then detected.

The configuration shown in Fig. 7(c) also tests FBDFs. If a forward-biased diode is defective, it may create a very high-volume resistor between vertical line and its attached horizontal lines and results in a zero on the horizontal output lines. K configurations are required to test all FBDFs, where K is the size of nanoblock (K × K). As a result, overall K test configurations are required to test FBDFs, sa0-H-PG, and stuck-open faults.

Fig. 8(a)–(c) shows configuration of nanoblocks required for detection of stuck-at-one fault on horizontal and vertical lines. As shown in Fig. 8(a), all inputs are connected to Gnd; hence, all horizontal output lines are zero. This configuration is used as PG. An output-line value becomes one in the presence of stuck-at-one fault on a horizontal line (sa1-H-PG). Stuck-at-one faults on horizontal and vertical lines of an RG block (sa1-H-RG and sa1-V-RG) can be detected using the configuration shown in Fig. 8(b). The test configuration shown in Fig. 8(c) is used to test sa1-V-PG and generates patterns for RG block.

The test configuration shown in Fig. 9 is used to test RBDFs in nanoblocks. All the diodes on the junctions are reverse-biased. To reverse bias the diodes, all the vertical input lines are connected to zero and the horizontal wires are connected to one. A RBDF may change the output-line value from zero to one or one to zero. Since zero is applied on vertical wires and one is applied on horizontal wires, this configuration must be used as RG. PGs generate required ones and zeros for RBDFs in RG to be tested. This test configuration requires two PGs in each TA; hence, TAs: TA51, TA52, and TA53 are used for this purpose.

Fig. 10(a) shows the required configuration to detect AND-bridging faults among vertical lines (ANDBF-V). A bridging fault between a line connected to Vdd and other vertical lines may change the vertical output, not connected to Vdd, from zero to one, and the output line connected to Vdd may change from one to zero. K configurations are used for detecting all faults on K vertical lines. Fig. 10(b) shows the required configuration to detect AND-bridging faults among horizontal lines (ANDBF-H). A bridging fault among two horizontal lines changes the output-line value. K configurations are required to test all AND-bridging faults among horizontal lines. Finally, Fig. 10(c) shows the required configuration for detecting OR-bridging faults (ORBF). An OR-bridging fault among vertical lines may change the output of vertical lines from zero to one. All three configurations must be used as RG blocks. Fig. 11(a)–(c) is used as PG to test ANDBF-H, ANDBF-V, and ORBF-H faults on PG, respectively.

B. Test Configurations for Detecting Faults in Switchblocks

In this section, we present the test configurations needed for testing switchblocks of the fabric. As shown in Fig. 2, a switchblock in the fabric is constructed from the intersection of inputs and outputs of four neighboring nanoblocks. This means that south and east outputs of two diagonally adjacent nanoblocks and north and west inputs of two other diagonally adjacent nanoblocks all together create a switchblock. The test configurations used for PG and RG to test stuck-at and bridging faults will completely cover them in switchblocks as well. This is because wires in a switchblock are the same wires of neighboring nanoblocks. To have a complete coverage of switchblock’s faults, additional test configurations must be designed to cover the remaining faults of switchblock, such as FBDFs and RBDFs. Fig. 12 shows two configurations required to detect FBDFs in diodes located on intersection of north and east wires of switchblock (FBDF-SB-NE) and diodes located on intersection of its west and south wires (FBDF-SB-WS). Inputs are applied from PG to SB so that north-east (NE) or west-south (WS) diodes are forward-biased, and the results are sent to RG for evaluation. Fig. 12(a) uses TA31 and TA32 and Fig. 12(b) uses TA41 and TA42 to detect all FBDF-SB-NE and FBDF-SB-WS faults in switchblocks.

Fig. 13 shows two other configurations needed to detect FBDFs on north–west (NW) and south–east (SE) diodes of switchblock. Since north and west lines are both inputs to switchblock, additional diodes must be configured to send the test results to RG [south lines are used for this purpose in
Fig. 10. Configuration for detecting (a) AND-bridging faults among vertical lines, (b) AND-bridging faults among horizontal lines, and (c) OR-bridging faults among vertical lines.

Fig. 11. PG for detecting (a) ANDBF-V, (b) ANDBF-H, and (c) ORBF-H faults in RGs.

Fig. 12. (a) Test configuration for testing FBDFs on north–east diodes (FBDF-SB-NE). (b) Test configuration for testing FBDFs on west–south diodes (FBDF-SB-WS).

Fig. 13. (a) Test configuration for testing FBDFs on north–west diodes (FBDF-SB-NW). (b) Test configuration for testing FBDFs on south–east diodes (FBDF-SB-SE).

Fig. 14 shows two more configurations to test RBDFs on north–east and west–south diodes of switchblocks. These two use TA31-TA32 and TA41-TA42, respectively. Finally, Fig. 15 depicts the configurations required to test north–west and south–east diodes of switchblocks for RBDFs. These two use
TEHRANIPOOR AND RAD: BUILT-IN SELF-TEST AND RECOVERY PROCEDURES 951

Fig. 14. (a) Test configuration for testing RBDFs on north–east diodes (RBDF-SB-NE). (b) Test configuration for testing RBDFs on west–south diodes (RBDF-SB-WS).

Fig. 15. (a) Test configuration for testing RBDFs on north–west diodes (RBDF-SB-NW). (b) Test configuration for testing RBDFs on south–east diodes (RBDF-SB-SE).

TA21, TA22, and TA11, TA12, respectively. The idea behind these TAs is just the same as what is done in the previous parts, apply inputs (through PG) so that diodes are reversed-biased and send the results out to RG. These four test configurations cover RBDFs of switchblocks.

C. Number of Test Configurations

Since we use a test group containing PG and RG, we target one fault in PG and another in RG (and usually bridging or stuck-at faults in the switchblock between them). If the output of RG is not equal to the fault-free one, a fault is detected. Fig. 16 lists the required test configurations to cover all stuck-at, ANDBF, and ORBF faults in nanoblocks and switchblocks. The first column shows that which TA is used. The second and third columns present the test configurations used in that TA for PG and RG, respectively. Detected faults in nanoblocks are shown in the fourth column, and the fifth column shows faults detected in switchblocks. Number of test configurations used for a TA, number of architectures, and the total number of configurations are listed in columns six, seven, and eight, respectively.

Fig. 17 shows the same information for detecting FBDFs and RBDFs on the nanoblocks, and finally, Fig. 18 lists the configurations for detecting FBDFs and RBDFs on switchblocks. The main objective at this point is to minimize the total number of required test configurations. Assume that we use seven TAs: {TA31; TA32; TA41; TA42; TA51; TA52; and TA53} to test a nanofabric. As shown in Fig. 16, when using TA31, configurations shown in Fig. 7(a) and (b) are used. Configuration Fig. 7(a), used as PG, detects stuck-at-zero faults on vertical lines (sa0-V-PG), and configuration Fig. 7(b), used as RG, detects stuck-at-zero faults on horizontal lines (sa0-H-RG). The number of configurations required to test all these faults, as shown in the last column, is two since both TA31 and TA32 are applied.

The TA is chosen based on the test configurations used to detect a particular fault. For example, to test sa0-V-PG faults, test configuration Fig. 7(a) generates one on its vertical line (south), because of which, we must use TA31 to transfer the pattern from south side of PG and send it to the east side of RG. Therefore, we need to use an RG like Fig. 7(b).

As shown in Fig. 17, configurations Fig. 7(b) and (c) are used in TA41 and TA42 to test FBDFs in nanoblocks. Overall, $K$ configurations are required with each TA. To test RBDFs, one configuration is used for three TAs, TA51, TA52, and TA53.

Based on these figures, the required number of configurations is

\[
\text{Number of configs for stuck at and bridging faults} = 8K + 8
\]
\[
\text{Number of configs for FBDFs} = 2K + 3
\]
\[
\text{Number of configs for RBDFs} = 8K + 8
\]
\[
\text{Total number of configs} = 18K + 19.
\]
Fig. 16. Number of test configurations required to test all the stuck-at and bridging faults in nanoblocks and switchblocks.

Fig. 17. Number of test configurations required to test all the FBDF and RBDF faults in nanoblocks.

(18K + 19) is the required number of configurations to achieve complete fault coverage for the faults listed in our fault model for test groups with nanoblocks and switchblocks of size K. (18K + 19) also indicates the total number of times a programming device configures the fabric under test. Next, we estimate the total number of switches to be configured in BIST. Let us assume that \(N_{\text{sw_config}}\) is the average number of switches to be configured in each test group for each test session. Thus, \((18K + 19) \times N_{\text{sw_config}}\) will be the total number of switches to be configured during BIST in each test group.

Suppose the fabric has clusters of size \(N\), an \(M \times M\) array of clusters \(M^2\) clusters) and \(N^2/4\) test groups in each cluster (see Fig. 1). Therefore, the total number of switch configurations in the fabric during BIST is calculated by \((M^2 \times N^2/4 \times (18K + 19) \times N_{\text{sw_config}})\). Under the same conditions, [21] reports \((4K^2 + 4K + 6)\) configurations for each test group.

Since test groups in [21] contain three nanoblocks, thus, the total number of test groups in a cluster will be \((N^2/6)\). Hence, the total number of switch configurations will be \((M^2 \times N^2/6 \times (4K^2 + 4K + 6) \times N_{\text{sw_config}})\), where \(N_{\text{sw_config}}\) in [21] is greater than \(N_{\text{sw_config}}\) in our method since the test groups are larger. It can be expected that \(N^2/6 \times N_{\text{sw_config}} \approx N^2/4 \times N_{\text{sw_config}}\). Therefore, the total number of switches to be configured using [21] is significantly larger compared to our technique due to \(4K^2\) factor in the number of required configurations for each test group in [21].

D. Time Analysis for BIST

Test time \(T\) for reconfigurable devices is constituted from three main components, namely: 1) configuration time \(T_{\text{CT}}\); 2) test-application time \(T_{\text{AP}}\); and 3) response-readback time.
\(T_{\text{RBT}}\), i.e., \(T = T_{\text{CT}} + T_{\text{AP}} + T_{\text{RBT}}\). In this section, we compare our technique with [21]. Test-application time \(T_{\text{AP}}\) for both techniques is very short, independent of the fabric size, and negligible when compared to \(T_{\text{CT}}\) and \(T_{\text{RBT}}\). Once the test groups are configured, the test pattern will be generated and applied in parallel in all test groups of the fabric.

Considering similar architecture for both techniques, \(T_{\text{RBT}}\) of our technique will be larger since: 1) more number of test groups \((N^2/4)\) must be readout (compared to \((N^2/6)\) in [21]) and 2) \(K\) output lines of RGs are readout compared to one in [21]. Note that our technique’s ability in finding the location of faults is promising, since more output lines are readout. On the other hand, we use lower number of test configurations. We can argue, based on the previous discussion, that our technique’s \(T_{\text{RBT}}\) will not be significantly higher than that of [21].

Accurate analysis of test time for our technique and [21] would require detailed information about the routing and interconnect architecture of the nanofabric. \(T_{\text{CT}}\) depends on the programming device’s configuration capability and bandwidth. Let us assume that programming device can configure \(C\) switches in one clock cycle. Thus, the total number of clock cycles required for BIST procedure is given by

\[
(M^2 \times N^2/4 \times (18K + 19) \times N_{\text{sw-config}}) / C.
\]

As shown, the total number of clock cycles is proportional to the total number of test group’s configurations \([(18K + 19)\) versus \((4K^2 + 4K + 6)\)]. This demonstrates that our technique, due to using less number of test groups, offers lower \(T_{\text{CT}}\).

Similar analysis can be performed for RI procedure with the difference that RI does not configure, test, and read all the fabric blocks. In RI, only susceptible and recoverable blocks are considered.

V. Recovery Analysis

If a test group is fault-free after applying all the test configurations, all the components in that test group are identified as fault free. But, if the test group is faulty, at least one component (PG, RG, or switchblock) in that test group is faulty. The PG and RG used in this test group are also used in other test groups in different architectures. Assume that each test group is represented as \((\text{NB}_{ij}, \text{SB}_{pq}, \text{and} \text{NB}_{kl})\), where \(\text{NB}_{ij}\) denotes a nanoblock (PG) located in row \(i\) and column \(j\), \(\text{NB}_{kl}\) is RG located in \(k\)th row and \(l\)th column, and \(\text{SB}_{pq}\) is the switchblock located in row \(p\) and column \(q\). For instance, Fig. 19(a) shows that \(\text{NB}_{22}\) and \(\text{NB}_{33}\) are used in TA32 and TA42 as PG and RG, respectively. As shown, \(\text{NB}_{22}\) is also used as RG in TA31, TA41, and TA51 and \(\text{NB}_{33}\) is used as PG in TA31 and TA41 for the RG block located at row four and column four (\(\text{NB}_{44}\)).

Fig. 19(b) shows a case where the test group (\(\text{NB}_{22},\ \text{SB}_{32}, \text{and} \text{NB}_{33}\)) is faulty only when testing sa0-V-PG and sa0-H-RG using TA32 and the test groups (\(\text{NB}_{11}, \text{SB}_{12}, \text{and} \text{NB}_{22}\)) and (\(\text{NB}_{33}, \text{SB}_{34}, \text{and} \text{NB}_{44}\)) are fault free for all the test configurations. If one has this information about the fault-free status of these blocks, the only component in test group (\(\text{NB}_{22}, \text{SB}_{32}, \text{and} \text{NB}_{33}\)) that has an unknown status is \(\text{SB}_{32}\). Based on the information obtained from previous tests that sa0-V-PG in \(\text{NB}_{22}\) is tested when testing test group (\(\text{NB}_{11},\ \text{SB}_{12}, \text{and} \text{NB}_{22}\)) and sa0-H-RG is tested when testing test group (\(\text{NB}_{33}, \text{SB}_{34}, \text{and} \text{NB}_{44}\)), it is confirmed that \(\text{NB}_{22}\) and \(\text{NB}_{33}\) are fault-free and \(\text{SB}_{32}\) is faulty. This case is the simplest one, but if the neighboring test groups are not fault free, then an extra process is needed to clarify which components of the susceptible test group are faulty or fault free.

We continue with the example and change our assumption about fault-free status of neighboring test groups [Fig. 19(c)]. We assume that previous tests has confirmed that nanoblocks \(\text{NB}_{31}\) and \(\text{NB}_{35}\) are not susceptible for horizontal faults, \(\text{NB}_{42}\) has no vertical faults, switchblock \(\text{SB}_{12}\) is not susceptible to

<table>
<thead>
<tr>
<th>Test Architecture</th>
<th>Configuration</th>
<th>SwitchBlock Faults</th>
<th># of Configs in a TA</th>
<th># of Archs</th>
<th>Total # of Configs</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA31 and TA32</td>
<td>11(a)</td>
<td>7(b) 12(a)</td>
<td>FBDF-NE</td>
<td>K</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>11(b)</td>
<td>7(b) 14(a)</td>
<td>RBDF-NE</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>TA41 and TA42</td>
<td>11(c)</td>
<td>8(b) 12(b)</td>
<td>FBDF-WS</td>
<td>K</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>11(c)</td>
<td>8(b) 14(b)</td>
<td>RBDF-WS</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>TA21 and TA22</td>
<td>11(b)</td>
<td>7(b) 13(a)</td>
<td>FBDF-NW</td>
<td>K</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>11(b)</td>
<td>7(b) 15(a)</td>
<td>RBDF-NW</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>TA11 and TA12</td>
<td>11(a)</td>
<td>7(b) 13(b)</td>
<td>FBDF-SE</td>
<td>K</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>11(a)</td>
<td>7(b) 15(b)</td>
<td>RBDF-SE</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Total number of configs for FBDFs and RBDFs in switchBlocks: \(8K+8\)
have WS (west–south) faults, and SB$_{34}$ is not susceptible to have WE (west–east) faults. To check the status of each of the components in test group (NB$_{22}$, SB$_{32}$, and NB$_{33}$), some more configurations must be applied. First, SB$_{12}$ and NB$_{22}$ should be configured in TA21. If the results of the applied test show no faults, NB$_{22}$ has no vertical faults because SB$_{12}$ has already shown itself free from south faults, if there is a fault, it must be in NB$_{22}$. The next step is to configure (NB$_{33}$, SB$_{34}$, and NB$_{35}$) as a test group. If the result shows no faults then NB$_{33}$ is fault-free, but if there is a fault, it must be in NB$_{33}$ because the other two blocks in the test group are confirmed to have no horizontal faults.

Diagnosing SB$_{32}$ for east faults is possible, if its eastern nanoblock (NB$_{33}$) has no horizontal faults. In this case, test group (NB$_{31}$, SB$_{32}$, and NB$_{33}$) can be configured in TA11, and if the result shows no faults, it means that SB$_{32}$ has no east faults. This is the same for testing SB$_{32}$ for north faults. Test group (NB$_{22}$, SB$_{32}$, and NB$_{42}$) should be configured in TA22, and if the result has no faults, SB$_{32}$ must be free of north faults. If the north neighbor of SB$_{32}$ is still susceptible to have vertical faults, then, there is no way to confirm the status of this switchblock for north faults.

If the total number of obtained fault-free test blocks after completing BIST is higher than a limit ($R_{\text{limit}}$) defined by the manufacturer, then, the chip is ready for shipping. Otherwise, a RI procedure needs to be performed to increase the recovery to $R_{\text{limit}}$. To perform this procedure, we assume that the external or onchip tester knows the FTGs and the fault that made the test group faulty. This information is obtained by running the BIST procedure proposed in previous section. The RI procedure will reconfigure the FTGs and/or their surrounding nanoblocks to target some particular fault(s). This results in increasing the number of test configurations, and the increase depends on the required $R_{\text{limit}}$. The higher the $R_{\text{limit}}$, the more number of test configurations may be required.

One advantage of using our additional TAs (e.g., TA11, TA12, TA21, and TA22) is that they provide a capability of finding the location of faulty component(s) in a defective test group. One may see that TA11, TA12, TA21, and TA22 do not increase the fault coverage. These architectures are solely used for diagnosis and in RI procedure.

An algorithm is required to analyze and find the minimum number of additional TAs/configurations to find the faulty or fault-free components in each FTG. The algorithm needs to consider the results obtained from applying TA31, TA32, TA41, TA42, TA51, TA52, and TA53 during BIST procedure to the nanofabric under test as input and extract a minimum set of additional TAs and configurations to recover the susceptible test groups and improve the recovery. This is performed by our proposed RI procedure, which is shown in Fig. 20. The core of this procedure is a linear search in the defect map. It identifies each of the defective blocks and checks the status of its neighbor blocks. If a test group can be constructed using the defective block and its fault-free neighbors, then that test group will be scheduled to be applied to the blocks. When all possible recovery TAs are extracted from the defect map, the BIST procedure will be executed for set of recovery architectures and configurations [RUN_BIST(RTA)]. Test results should be used to update the defect map [UPDATE_DM()].

Computational complexity for searching the defect map of nanofabric and identifying TAs that can be implemented around each susceptible block to recover it is $O([M^2N^2])$. It shows that this search can be performed in an acceptable time. Timing requirements of configuring the TAs required for recovery of the blocks and reading the test results will depend on the configuration time and routing resources of the fabric (as discussed in Section IV-D). If these resources provide the possibility of parallel configuration of these TAs, recovery procedure can be performed much faster. Total number of iterations required for the RI procedure depends on the test-time limitations or the acceptable number of fault-free blocks detected in the fabric. Our simulation results, shown in the next section, demonstrate that a low number of iterations considerably improves the recovery and takes it to acceptable range.

VI. SIMULATION METHOD AND RESULTS

A C++ model is created for the cluster and faults are randomly injected and BIST and RI procedures are performed. **Phase 1** In this phase, an array model of the fabric is created and faults are randomly injected to it. Each block (nanoblock/switchblock) is modeled as an array element.
Phase 2) In this phase, using IA and applying BIST procedure to the cluster under test, a second array is created, which contains the information of FTGs and the type of faults. Result of this phase models the output of test process in the real world and is used as input to the next step, which is the RI procedure. We refer to this array as aftertest array (AA).

Phase 3) RI procedure, as described in previous section, will find those blocks, which are not faulty but are marked as susceptible to fault during BIST. This process is applied to AA and results are stored in another array, which is the output of RI procedure. We refer to this array as recovered array (RA).

Phase 4) Analysis of the results is done by comparing AA with IA that shows recovery before running RI procedure and comparing RA with IA that shows recovery after running the procedure.

Phase 5) Repeat the steps described in phase 3) and phase 4) for a number of iterations, each time transferring the resulted RA to AA as input to the RI procedure and receiving the new RA as output. Number of iterations can be either defined by user, or the RI continues until there is no improvement in recovery.

These iterations are used to analyze the RI in each step of procedure. Fig. 21 is a schematic view of the simulation results for a 15 × 15 cluster with 5% defect density. IA, AA, and resulted RA are shown. The five arrays shows that BIST procedure marks some fault-free blocks as faulty and RI procedure identifies some of these fault-free blocks after each iteration.

To analyze the functionality of proposed RI procedure, simulation process is repeated for different cluster sizes and different defect densities, and the results are summarized in

---

```
# Define DM (Defect Map) // Database used to store the fault locations and their types.
# Define RTA (Recovery TAs) // Database used to store location and type of TAs used for recovery.
# Define M (Fabric size) // Fabric is a M × M array of clusters.
# Define N (Cluster size) // Clusters are N × N arrays of NanoBlocks and switchBlocks.
# Define K (Block size) // NanoBlocks are K × K arrays of nanowires.

RUN BIST Procedure (TA);
UPDATE_DM(); // Location and type of faults are stored in the defect map.

// RI Procedure //
DO {   
  for (i = 1; i <= M^2; i++) {
    for (j = 1; j <= N; j++) {
      for (l = 1; l <= N; l++) { // For each cluster, check the status of each NB or SB.
        if (DM(i, l) == Faulty) { // If Defect map shows that there is a fault in block(i, j)
          for (p = i - 2; p <= i + 2; p++)
            for (q = j - 2; q <= j + 2; q++) { // Search Neighbors of the faulty block
              if (DM(p, q) == Faulty) { // That TA is enabled using block(p, q) and block(j, j))
                RTA ← (TA including block(p, q) and block(j, j)) // That TA is stored in the RTA database.
              }
            }
          }
        }
      }
    }
  }
}
for (all TAs in RTA)
RUN_BIST_PROCEDURE (RTA);
UPDATE_DM();
} WHILE (iteration limit for the procedure reached or number of remaining blocks in DM is acceptable)
```

---

Fig. 20. RI procedure.

Fig. 21. (a) IA is the exact defect map for a 15 × 15 array with 5% defect density. (b) AA is the resulted defect map from BIST procedure. (c), (d), and (e) RA: resulted defect maps from RI procedure after first, second, and third iterations.
Figs. 22–24. These graphs show the amount of increase in recovery after applying RI procedure for fabrics with different cluster sizes (20 × 20, 50 × 50, and 100 × 100 blocks) with different defect densities (5%, 10%, 15%, and 20%) for a number of iterations. As shown, RI procedure significantly improves recovery for different fabrics and defect densities.

As shown in Figs. 22–24, when defect density is low (≤ 5%), the BIST procedure results in a reasonably high recovery (≥ 60%). In this case, RI procedure increases the recovery very quickly because the number of FTGs is low. As the defect rate increases, the recovery resulted from BIST procedure decreases and RI procedure also needs more iterations to improve recovery. It should be also noted that for high defect densities, some of the fault-free blocks will become nonrecoverable. Therefore, the recovery cannot increase significantly even if the number of RI procedure iterations further increases. As shown in Figs. 22–24, the starting point of curves for different sized fabrics is almost the same for a specific defect density (for example 40% for 20 × 20, 50 × 50, and 100 × 100 fabrics in 10% defects, compare Figs. 21–23). Therefore, the BIST procedure results in the same amount of recovery for different sizes of clusters. But as the figures illustrate, the amount of increase in recovery for an iteration of RI procedure reduces as the cluster size increases. Therefore, more iterations will be required for fabrics with larger number of blocks in each cluster.

Simulations were performed on arrays of clusters to estimate the time of RI procedure for real size nanofabrics. These simulations were performed on a 2.8-GHz Pentium 4 CPU with 1-GB RAM. Fig. 25 shows the elapsed CPU time in each case and the number of available molecular switches in the fabric array. The simulation time is reported for various fabric sizes (M = 50, 100, and 200) and cluster sizes (N = 20, 40, and 80). The results are reported for RI time (or CPU run time) and number of switches. The RI time is reported for the first iteration only. The next iterations time will significantly drops, since the number of remaining blocks to be recovered significantly reduces.

We acknowledge that the simulation procedure employed in this paper does not take the actual test time into consideration, since the tester and programming device’s capabilities and the nanoarchitecture are not well understood and clear yet.

VII. DISCUSSION

In this paper, we presented BIST and RI procedures for island-style nanofabrics. Some parts of the techniques proposed in this paper are general (architecture-independent) and can be expanded to other architectures [24], [26]. Test configurations discussed for detecting faults in nanoblocks and switchblocks

<table>
<thead>
<tr>
<th>Cluster Size (N)</th>
<th>Fabric Size(M)</th>
<th>RI time (sec)</th>
<th># of Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>20</td>
<td>0.7</td>
<td>2.9</td>
<td>11.8</td>
</tr>
<tr>
<td>40</td>
<td>3.1</td>
<td>12.1</td>
<td>48.6</td>
</tr>
<tr>
<td>80</td>
<td>14.4</td>
<td>62.9</td>
<td>250.7</td>
</tr>
<tr>
<td></td>
<td>First RI iteration time and # of Molecular Switches in the Fabric (K = 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1.6x10^{11}</td>
<td>6.4x10^{11}</td>
<td>2.5x10^{12}</td>
</tr>
<tr>
<td>40</td>
<td>4x10^{10}</td>
<td>1.6x10^{11}</td>
<td>6.4x10^{11}</td>
</tr>
<tr>
<td>80</td>
<td>1x10^{10}</td>
<td>6.4x10^{11}</td>
<td>250.7</td>
</tr>
</tbody>
</table>

Fig. 25. CPU elapsed time for RI procedure performed on different fabrics with different cluster sizes. Defect density is set to 5%.
are general and can be used in all architectures that are based on nanowire crossbars. But our TAs are designed based on [1] and may not be suitable in other architectures like [24] and [26]. The concept of RI procedure is also expandable to other architectures even though its implementation might be architecture-dependent.

Accurate test methods should be devised based on precise fault models derived from manufacturing and physical properties of the fabrication technology. Due to high defect densities expected in nanoscale devices, consideration of multiple faults may be required in future proposed test methods. Considering multiple faults will make the test procedure more complex. In this paper, the test procedure is based on detection of single faults. However, our investigation indicated that detection of multiple faults is possible and they can be detected in most cases. Since our technique relies on reading the output of RGs, multiple faults not located on the same wire can be detected because the faults effect will be observed on different lines of the RG block and the faults will not mask each other. If multiple faults occur on the same lines, they will mask each other and their detection may not be guaranteed. However, since multiple test configurations are applied to each test group, it can be expected that even if some faults mask each other in one test configuration, they may not do so in other configurations. In other words, the probability that multiple faults mask each other in all test configurations applied to the test groups will be very small. Therefore, the proposed technique works efficiently in detecting multiple faults, but high-fault coverage is not guaranteed. It may also be possible to add a set of test configurations to provide higher coverage of multiple faults.

Precise comparison of previous methods [19]–[21] with the technique proposed here, requires uniform assumptions about the routing and interconnect structure of nanofabric and uniform implementation environments for all these techniques. The method proposed in [19] uses test circuits that can count the number of detected faults (LFSRs or Counters) to assign defect probabilities to block of the fabric. Implementation details of these test circuits is dependent on the routing resources of the fabric. The technique proposed in [20] requires the clusters to be capable of saving defect information. This capability requires detailed design of a memory structure for nanoblocks and its interconnections with other resources of the fabric. The technique proposed in [21] is conceptually close to test method proposed in this paper. However, the main differences are: 1) test group; 2) TAs; 3) test configurations for switchblocks and some of the TAs for nanoblocks; and 4) RI procedure. Test groups designed in this paper are smaller (finer granularity) and TAs and configurations support testing both nanoblocks and switchblocks. Finer granularity for the test, as we discussed before, results in higher recovery. Implementation of all these methods relies on availability of rich interconnect and configuration circuitry for configuring the test groups and reading test results. The configuration-circuitry area overhead, as in [21] and our technique, to support access to all the nanoblocks in a cluster can be large, but it provides higher accessibility and testability, which result in higher recovery. Comparing [21] and our technique shows that our technique can provide higher diagnosability since all the outputs of RG blocks are readout by the tester for evaluation.

VIII. CONCLUSION

In this paper, we presented a BIST procedure to test nanoblocks and switchblocks in a fine-grained nanofabric. Various fault types were considered, and a set of test configurations and TAs for detecting the faults were proposed. The BIST procedure tests the fabric step-by-step through applying these architectures and configurations. During this procedure, blocks are tested in parallel and this reduces the test time significantly. BIST procedure partitions the cluster under test into small test groups, which include PG, RG, and associated switchblock. Using small test groups increases the diagnosability and recovery. The proposed BIST procedure is simple, efficient, and quick, since it uses a very small number of test configurations and it detects all faults listed in the proposed fault model. RI procedure was presented to test and diagnose the test groups that are susceptible to be faulty and to find the fault-free components in that test group. A simulation program was also designed, and the simulation results were presented to show the effectiveness of proposed RI procedure.
Mohammad Tehranipoor (S’02–M’04) received the B.Sc. degree from Amirkabir University of Technology (Tehran Polytechnic University), Tehran, Iran, in 1997, the M.Sc. degree from the University of Tehran, Tehran, in 2000, and the Ph.D. degree from the University of Texas at Dallas, in 2004, all in electrical engineering.

From 2004 to 2006, he was a member of the faculty of the Department of Computer Science and Electrical Engineering, University of Maryland in Baltimore County (UMBC). He is currently an Assistant Professor of electrical and computer engineering at the Department of Electrical and Computer Engineering, University of Connecticut, Storrs. He has published more than 60 journal articles and refereed conference papers in the area of VLSI design and test. His current research projects include computer-aided design and test for CMOS VLSI designs and emerging nanoscale devices, design-for-testability, at-speed test, and secure design.

Dr. Tehranipoor is a member of the IEEE Computer Society, the Association for Computing Machinery (ACM), and the ACM Special Interest Group on Design Automation. He is a recipient of the Best Paper Award at the 2005 VLSI Test Symposium and the Best Paper Candidate at the 2006 Design Automation Conference. He has served on the program committee of several workshops and conferences in addition to serving as Session Chair in many technical events. He is a Guest Editor of the Journal of Electronic Testing: Theory and Applications on “Test and Defect Tolerance for Nanoscale Devices” and a Co-Guest Editor of the IEEE Design and Test of Computers on “IR-drop and Power Supply Noise Effects on Very Deep-Submicron Designs.” He will be serving as Co-Program Chair of the 2008 International Defect and Fault Tolerance in VLSI Systems and Program Chair of the 2007 IEEE Workshop on Defect-Based Testing.

Reza M. P. Rad (S’05) received the B.Sc. degree in electrical engineering and M.Sc. degree in computer engineering from the University of Tehran, Tehran, Iran, in 1999 and 2001, respectively. He is currently working toward a Ph.D. degree in computer engineering at the University of Maryland Baltimore County.

He is a Research Assistant in computer engineering with the University of Maryland Baltimore County. His research interests include design, testing, and defect and fault-tolerance issues in digital systems, reconfigurable structures, and nanotechnologies.