On-Line Fault Detection for Bus-Based Field Programmable Gate Arrays

Nathan R. Shnidman, William H. Mangione-Smith, *Member, IEEE,* and Miodrag Potkonjak

***Abstract*— We introduce a technique for on-line built-in self-testing (BIST) of bus-based field programmable gate arrays (FPGA's). This system detects deviations from the intended functionality of an FPGA without using special-purpose hardware, hardware external to the device, and without interrupting system operation. Such a system would be useful for mission-critical applications with resource constraints. The system solves these problems through an on-line fault scanning methodology. A device's internal resources are configured to test for faults. Testing scans across an FPGA, checking a section at a time. Simulation on a model FPGA supports the viability and effectiveness of such a system.**

***Index Terms*— Digital system fault tolerance, field programmable gate arryas, self-testing.**

I. Introduction

THE most common type of programmable logic device available today is the field programmable gate array (FPGA), which are arrays of programmable gates and routing resources. The use of FPGA's in consumer applications, in addition to their more traditional use in logic emulation systems, prototyping, and low-volume applications, has increased due to recent advances in technology. Unfortunately, many of the trends that make newer FPGA's more appealing and affordable also make them less reliable. For example, high-density programmable devices are made more susceptible to gamma particle radiation by smaller feature sizes, and the corresponding lower threshold voltages. Also, larger die sizes make interference from such radiation much more likely [1]. Extensive terrestrial efforts to accurately model the rate of such soft faults indicate high variance (several orders of magnitude) depending on factors such seasonal solar activity, altitude, latitude, device technology, and device materials. Even for the same chip from the same manufacturer, variations by a factor higher than 200 are not uncommon [1]. Experiments indicate that in FPGA-like devices at an altitude of 20 km, error rates significantly higher than once per 1000 h are common [2]. A second class of faults is related to manufacturing imperfections. These defects are not large enough to impact initial testing, but after a longer period of operation they become exposed. Imperfections of this sort can become manifest as either stuck-at faults or as transient faults (which are not addressed here). Design errors can also cause a device to stop functioning in response to rare sequences of inputs (e.g., due to a power-density surge in a small part of design).

Ironically, while hardware reuse is a primary reason for utilizing an FPGA, external hardware for fault testing and tolerance of FPGA's often requires large amounts of additional system resources. Implementing these tasks external to the FPGA requires that the functionality of the device be interrupted in order to detect and address faults. This approach not only results in system functionality being interrupted periodically, but also in fault testing only occurring periodically. Thus, the time between a fault occurring and being detected could be significant. Most importantly, soft faults in the volatile memory of an FPGA will not be detected by such a system without adding a time consuming read-back step.

Making use of resources internal to the FPGA to implement a fault detection system, however, avoids these problems. A portion of the device resources are set aside to perform the fault handling, but fewer system resources are consumed than with an external fault monitor. An internal fault monitor can also run in the background on an FPGA that supports partial reconfiguration. Such a fault monitor could run continuously, while not impeding device functionality. Another advantage to an internal fault monitor is that it conserves limited pin resources and avoids the relatively slow process of transferring information off-chip through the pins. This approach allows for rapid detection of both hard and soft faults.

The resources needed to perform fault testing can be kept to a minimum by using a fault scanning methodology. Only a small section of the FPGA is tested at a time, but testing can scan across the FPGA assuring that the entire FPGA will be tested eventually. It is also necessary to take the resources being tested off-line (but not the entire system) to perform the fault detection tests. By testing only a small portion of the FPGA at a time the approach allows fault testing to occur without interrupting functionality.

A. FPGA Architectures

FPGA's have cell-like structures. The cell is used to implement the functionality of a number of gates, and it also commonly contains a small amount of memory. The number of gates per cell is dependent on the FPGA architecture, but usually ranges from one to six gates. Part of the programmability of the FPGA comes from the fact that the designer can

Manuscript received July 30, 1997; revised December 18, 1997.

N. R. Shnidman is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: naters@mtl.mit.edu).

W. H. Mangione-Smith is with the Department of Electrical Engineering, The University of California, Los Angeles, CA 90095-1594 USA (e-mail: billms@ucla.edu).

M. Potkonjak is with the Department of Computer Science, The University of California, Los Angeles, CA 90095-1596 (e-mail: miodrag@cs.ucla.edu).

Publisher Item Identifier S 1063-8210(98)07562-3.

change the actual type of gates implemented by each cell. In addition, the user determines if the combinational logic section of the cell or the memory section of the cell, or both, are used.

There are two basic models for the combinational logic portion of cells. One is called *island-based*, the other is called *fine-grained* [3]. The island-based FPGA uses one or more look-up-tables (LUT's) per cell to provide the functionality of gates. These cells typically have four or more inputs. In contrast, the cells of a fine-grained FPGA usually have only two inputs. The small number of inputs often allows logic in a fine-grained cell to be implemented with multiplexers [4].

There are also two types of programmable interconnect. One type of interconnect involves point-to-point or segmented buses. This model has wires of varying lengths placed horizontally and vertically throughout the FPGA, with switches connecting the pieces of interconnect. Programming such an FPGA requires a routing step, where an attempt is made to connect cells together using the least amount of interconnect. Since interconnect needs are unknown *a priori* in such a model, it is possible that designs exceed the amount of available interconnect. The Xilinx XC4000 family is an example of a segmented bus architecture.

The other form of interconnect is termed bus-based. This model involves long interconnect lines which span all (or a significant portion) of the chip. Connections between cells are made by writing to and reading from these buses. Bus-based interconnect tends to be slower than point-to-point due to increased wire capacitance. However, bus-based interconnect has the advantage of predictable timing, because the time to drive all signals is the same.

To illustrate a commercial FPGA design, consider the Flex10K by Altera [5]. The Flex10K uses static memory-based LUT's, with a single flip-flop. In addition to the basic cell structure, the Flex10K groups eight cells, or logic elements (LE's), into what is called a logic array block (LAB). These LAB's provide point-to-point interconnect on a small scale, for fast local communication. The LAB's are arrayed in a grid pattern, and are connected by chip-length buses. The place-and-route software for the Flex10K attempts to constrain designs to units that fit within a LAB, and buses make routing after placement trivial.

B. Motivational Example

An on-line internal fault detection system would be of particular use in space-based applications [6]. Limited resources such as volume, weight, and power make the use of FPGA's particularly appealing due to the opportunity to use time-sharing among the circuits to increase functional density. Furthermore, the use of additional hardware to perform fault testing and fault tolerance is particularly unappealing because of the resource constraints. Thus, traditional approaches such as triple modular redundancy [1] are tolerated, but at a great expense. Space-based systems are also subject to more operational interference from radiation and charged particles than terrestrial systems. As such, it is imperative that faults be located and addressed quickly.

A fault scanning system would address most of these concerns. By using resources internal to the FPGA the system avoids an increase in system resources. The transparent nature of the fault scanner also allows it to run continuously, thus providing quick detection of faults. If the fault tolerance mechanism discussed below is implemented, the fault scanner could even allow the FPGA to tolerate some faults and continue to function.

This paper presents multiple possible internal on-line fault scanning monitors for FPGA's, and simulations showing a proof-of-concept implementation. We will discuss the available design options and the simulation in Section II. This section will also address the specific FPGA used for the simulation and how it compares to current FPGA's. Section III will discuss previous work in the area. The basic fault scanning system is developed in Section IV. Alternative faults scanners will be presented in Section V, along with a discussion of the relative advantages of the various systems. Simulation results demonstrating the functionality of the fault scanner are considered in Section VI. Some concluding remarks are given in Section VII.

II. PRELIMINARIES

Memory faults are usually categorized into the groups: parametric and functional [7]. Parametric faults are related to design errors and include too low or too high output levels, insufficient fan-out driving capability, inadequate noise margin, and too short data retention interval. This type of fault is addressed by using proper design techniques.

There are three widely accepted RAM functional fault models: stuck-at-faults, coupling faults, and pattern-sensitive faults [7], [8]. Stuck-at faults correspond to situations where a particular memory cell is stuck-at-1 or -0. Coupling faults are related to manufacturing errors which cause one cell to change its state as a consequence of a state change in another cell. Pattern-sensitive faults are faults which alter the state of a memory cell as a consequence of a certain pattern of zeros and ones being stored or read from some other cells.

The initial manufacturing testing addresses the needs of the last two types of functional faults. After the FPGA design is configured, the only relevant faults are the stuck-at-faults which may occur as the consequence of particle radiation. Therefore, our in-field testing focuses on stuck-at-faults. Single-event upset (SEU) [9], [10] faults are also captured by the approach.

Even though the majority of the FPGA area is taken up by interconnect, it is reasonable to focus the fault detection system on the LUT's and flip-flops. These structures hold the programmable components of the device, and are much more likely to experience faults than metal interconnect wires. Furthermore, the fault detection system can only be expected to detect persistent faults, i.e., faults that will affect system operation until they are addressed. These types of faults are most likely to occur in the LUT's and flip-flops due the fact that both of these elements have state.

Another assumption is that configuration memory and flip-flops are fault free when the original configuration is loaded into the FPGA. Physical defects in the configuration elements, or errors in configuration data are not addressed by this testing

system. The ability to read-back the configuration data which has been loaded into the FPGA would help to catch such defects. A mechanism has been included, however, to help detect faults in the configuration elements and datapaths.

Additionally, the FPGA cells must be modified as described below. The global control signals must be maskable, so that they can target specific cells. The FPGA must be partially reconfigurable. That is, the FPGA must have the ability to change the functionality of only some cells, while leaving the remaining cells untouched. The ability to change the functionality of only a section of the FPGA is at the heart of this on-line fault system, and is absolutely necessary to implement the system. However, it is not necessary that the system present a partial configuration capability to the designer.

III. Related Work

A great deal of the related work falls into four general categories: built-in self-test (BIST), built-in self-repair (BISR), FPGA yield enhancement, and FPGA faults in space-based systems.

BIST and BISR methods for detecting and handling faults have been used extensively in memory designs [9], [11], [12], as well as in FPGA's in an off-line manner [13]–[19]. The FPGA BIST techniques mainly focus on reconfiguring the entire FPGA into set states in order to identify and then locate faults. While these techniques work well and are well suited for certain applications, they all require reconfiguration of the entire FPGA, and therefore interrupt FPGA function. BIST techniques can be combined with BISR and other fault tolerance methods [20], [21] to increase the robustness of FPGA's.

BIST and even BISR methods have been used to detect and handle fabrication faults in order to improve fabrication yields [22]–[25]. There has also been work done dealing specifically with the susceptibility of FPGA's to faults in space-based situations [26], [27].

None of the above work, however, deals with on-line fault detection. Some of these systems provide fault detection, using BIST, on FPGA's. The major difference between our fault scanner system and previous work is that with the new approach the FPGA need not be taken off-line before fault testing can occur, i.e., the functionality of the system is not interrupted for testing purposes.

Shombert and Siewiorek have developed a roving technique for fault-scanning of systolic arrays [28]. This approach involves taking a part of the array off-line for testing, while leaving the rest of the array on-line and functioning. While their earlier work set some of the groundwork for fault scanning, the research presented here develops the approach in the context of low-level reconfigurable hardware devices.

IV. Approach

We present a system overview, the basic algorithm, and a discussion of the general FPGA architecture used in testing in this section.

A. System Overview

The basis of the internal FPGA fault system is a scanning methodology. The system allocates a portion of the FPGA to fault testing. Testing is accomplished by sweeping the test functions across the entire FPGA. If the functionality of a small number of FPGA elements can be replicated on another portion of the FPGA, then those components can be taken off-line and tested for faults in a transparent manner (i.e., without interrupting functionality). The fault scanning system can then move on to another set of elements to copy and then test, moving through the whole FPGA systematically testing for faults.

B. Basic Algorithm

Built in fault testing for FPGA's requires that some of the resources of the FPGA be used for testing purposes. Allocating too many resources reduces the functionality of the FPGA, while allocating too few resources results in slow testing. The basic unit of testing is the column. Focusing on a column at a time allows for parallel testing of multiple cells, while not excessively constraining the FPGA functionality.

Our basic on-line testing algorithm reserves two columns of the FPGA for testing. These columns are completely transparent to the computer-aided design (CAD) tools and, thus, are not part of the visible device architecture. This approach alleviates the need to consider the columns when conducting place and route. One of these columns, the testing column (TC), contains the testing state machine. This state machine produces the control signals that implement testing. At the moment, a second state machine, which keeps track of the column being tested, is modeled as being off-chip and is assumed to be fault-free. For designs of 20 cells per column or larger, this second state machine could also be implemented on-chip. The output of this state machine is used to mask the global testing control signals such that they only affect a single column. The other reserved column is the free column (FC), which acts as buffer space during testing.

The on-line testing algorithm (Fig. 1) consists of three basic steps: copy, test, and move. In the copy step, a functional duplicate is made of the next column to be tested. Copying is done by writing the data from the configuration memory and the configuration flip-flop to the configuration data (CDATA) bus (Fig. 2). Data on the CDATA bus is written to the FC, thus, making a functional duplicate of the column to be tested. The configuration memory and the FC's LUT's are sequenced through all possible cell input vectors, for all cells in the FC in parallel, by connecting their inputs to a counter. This approach results in each column being in one of two states: functional or testing. Each column is off-line, i.e., supporting testing, for the time required to test the column and then copy configuration state back into the column (Table I). Further, each column is executing its programmed function while each of the other columns is tested in sequence. Note that the testing time for another column includes the time to copy a configuration out of the column and into the FC, although during this time the column itself is functional.

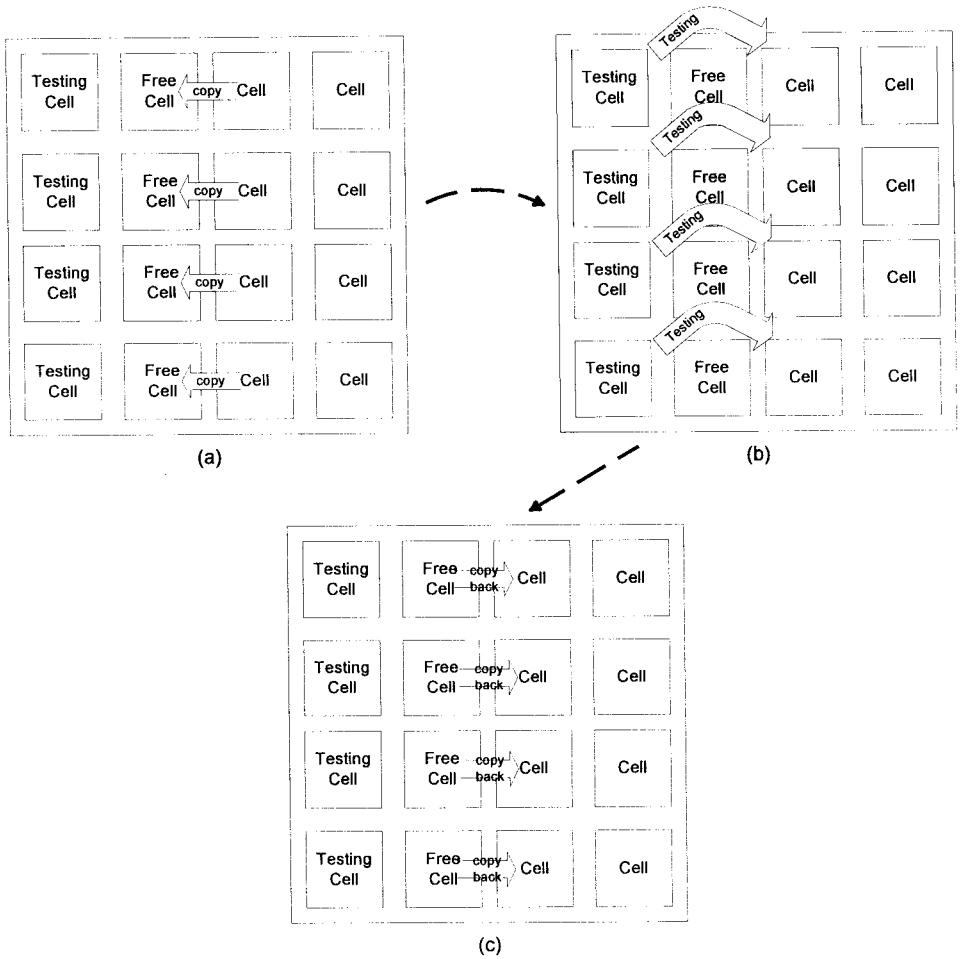


Fig. 1. Basic algorithm: copy functionality of column to be (a) test to free column, (b) test the column, and (c) copy the functionality back.

After the configuration memory is copied to the FC, the inputs and outputs of the FC are switched over to those of the cells in the column to be tested. The bus-based architecture of the FPGA makes this relatively simple. The FC cells tap the input buses of the cells being duplicated and the switch information in the configuration memory is used to set the outputs to drive the appropriate buses.

Next, the values in the flip-flops in the column to be tested are sent over the CDATA bus to the FC flip-flops. Write operations to a flip-flop always win over the copy in order to avoid stale data. If a write to a flip-flop of a cell in the column to be tested occurs during the copying process, the same value is also written into the flip-flop of the corresponding cell in the FC (because both flip-flops have the same inputs). Once the column to be tested has been copied, the FC outputs are turned on. Both columns are active with the same state, inputs, outputs, and functionality for a clock cycle in order to avoid glitches on the outputs. The outputs from the column to be tested are then tri-stated.

The next step is to perform the actual testing (Table II). The inputs to the column under test are connected to the output of the counter that is driving the configuration memory. The LUT and configuration memory are sequenced in parallel. Any difference between the output of the LUT and memory of any cell being tested indicates a fault. The outputs of the flip-flops and configuration flip-flops are then compared. Any difference between those also indicates a fault. These procedures test

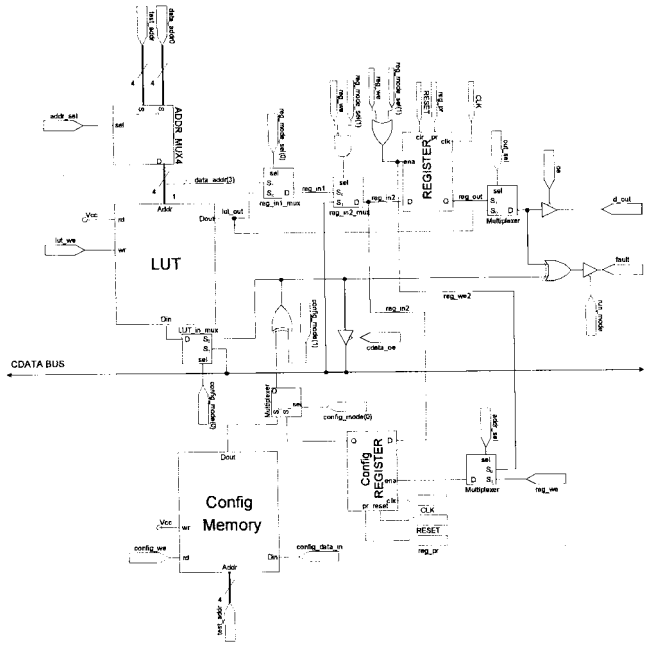


Fig. 2. Cell design for the fault scanning FPGA.

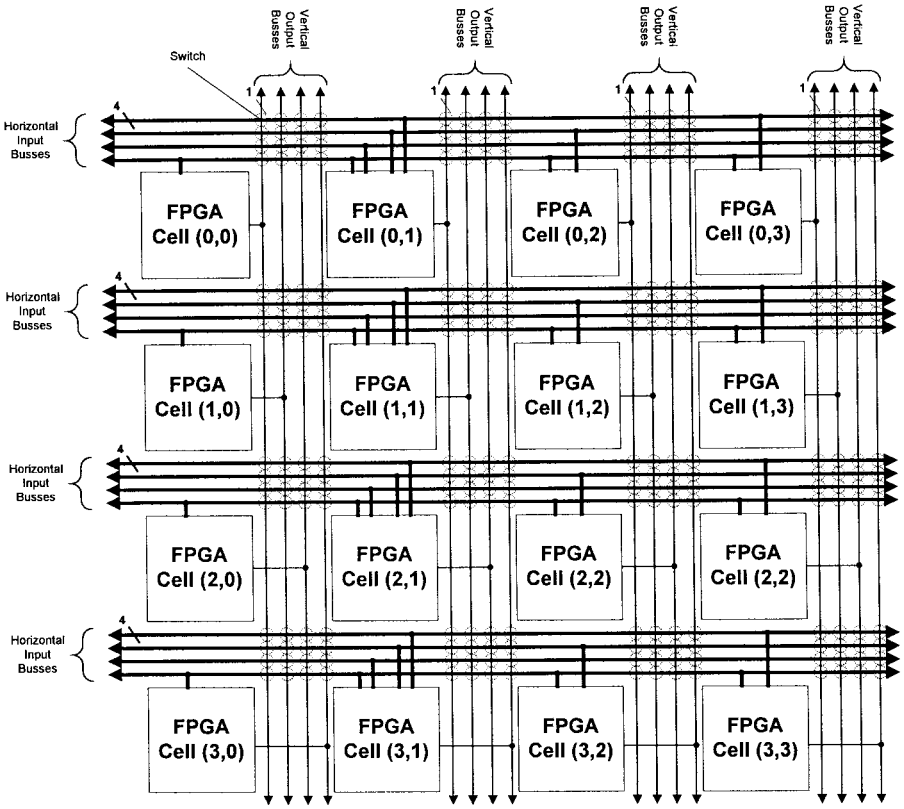


Fig. 3. FPGA cell array with interconnect.

for both SSF and SEU faults. It is necessary to have two copies of the correct cell functionality (i.e., the LUT and configuration memory and the two flip-flops) in order to detect SEU faults.

The next phase of testing is to write the inverse of the values in the configuration memories and configuration flip-flops to the LUT's and flip-flops. This allows the system to check for SSF faults. The outputs of the LUT are then compared to the inverted output of the configuration memory to test for differences. Any discrepancy between the two outputs indicates a fault. The outputs of the flip-flops are compared in a similar fashion to the inverted outputs of the configuration memories. Note that, while testing a cell for faults, it is possible to write to the cell's flip-flop independently of writing to the configuration flip-flop, although the converse is not true.

Using this approach, the LUT and flip-flops will be exhaustively tested for any SSF or SEU faults.

The final phase (move) involves transferring the original functionality back into the column that was just tested. This phase begins by writing the noninverted values in the configuration memories back into the LUT's. The inputs to the LUT's are then switched back to the correct buses. Next, the values stored in the FC flip-flops are written back to both the original flip-flops and configuration flip-flops. Writes take priority over copies, as with the previous copying to the FC flip-flops.

Finally, the outputs of the column that was just tested are turned on, and after waiting an extra clock cycle to avoid glitches, and the FC outputs are tri-stated. The algorithm is then applied to the column to the right of the column that was just tested. If there is no column to the right, testing begins at the leftmost column. This algorithm can be applied to both the TC and FC themselves, thus testing the entire FPGA.

C. System Architecture

The basic cell configuration can be seen in Fig. 2. The main functionality of the cell consists of a four-input one-output memory-based LUT and a single flip-flop. The LUT is a static memory with the four address bits used as addresses. The values in the LUT are stored during configuration of the FPGA. The cell has multiple possible modes that utilize the main cell elements differently: LUT alone, flip-flop alone, flip-flop controlled by LUT inputs, and LUT and flip-flop together. The simulated FGPA was loosely based upon the Altera Flex10K part. The main similarities consisted of the use of chip-wide buses as interconnect, and the use of four-input, one-output LUT's in the cells. The simulated FPGA, however, does not make use of grouped cells (LAB's) and dedicated memory blocks as does the Flex10K.

In addition to being functionality similar to a typical FPGA cell, the cell in Fig. 2 has extra elements that support fault testing. The major changes to the design are the addition of the configuration memory and the configuration flip-flop. In order to copy a cell, the LUT must be sequenced through all inputs, and the results written to the corresponding FC cell. The flip-flop data must also be copied. These elements allow the configuration data of a specific cell to be accessed

without affecting the operation of that cell. The configuration elements are also used in fault detection by comparing their output with the LUT and flip-flop outputs. During normal operation all writes to a flip-flop also write the same data to the configuration flip-flop, avoiding stale data.

The cells are arrayed in a grid pattern in the FPGA and are connected by device-length buses (Fig. 3). Horizontal buses carry the cell inputs, which are 4-bit wide, and the vertical buses carry the cell outputs, which are 1-bit wide. There are as many horizontal buses associated with each row of cells as there are cells in a row, and as many vertical buses as there are cells in a column.

Switches connect each vertical bus to every horizontal bus in the FPGA. For example, if the output of cell (1, 1) is to be used as the LSB and MSB inputs into cell (2, 3) the switches connecting the vertical output bus of cell (1, 1) to the first and fourth bits of the input bus to cell (2, 3) would be turned on. The state of these switches is set by the configuration data loaded into the FPGA. A copy of the state loaded into all of the switches at a juncture is also stored in the configuration memory at that juncture. So the state of all switches at the juncture of row three and column two are stored in the configuration memory of cell (2, 3). The switches have the ability to write their state value to the CDATA bus. This means that during the testing phase the state of each switch at a juncture can be compared with its intended value, which is stored in the configuration memory. Additionally, this capability allows switch settings to be copied to the FC during the copy phase.

In addition to the cell-to-cell connections, there are also global connections within the FPGA. Many of the global connections carry control signals to the cells. Global signals can be masked to affect only specific cells, or columns of cells, in addition to all cells.

Most of the global connections not used for control signals are used to carry configuration data, that is, data that specifies the values to be stored in the LUT and flip-flops. Of particular interest is the CDATA bus. There is a CDATA bus for each row in the FPGA. The bus is used to access the configuration data of a cell so that a copy of that cell can be made. This capability is necessary for on-line fault testing to occur.

The other configuration data global connections are used whenever a new configuration is input into the FPGA. It should be noted, however, that it is possible to eliminate these connections by simply tying them to the CDATA bus. This would provide the same functionality, but would theoretically slow down the reconfiguration process, as there is only one CDATA bus per row. However, since input-output (I/O) pins on an FPGA are usually limited, configuring all cells in parallel is generally not possible, thus making the elimination of the extra connections preferable.

The only other type of global connection is the fault bus (FB). Each row has an FB, which stretches the entire length of the FPGA. Each cell in a row is connected to the FB, but the fault output of each these cells is tri-stated if that cell is not currently being tested. The FB is used to indicate the presence of a testing failure.

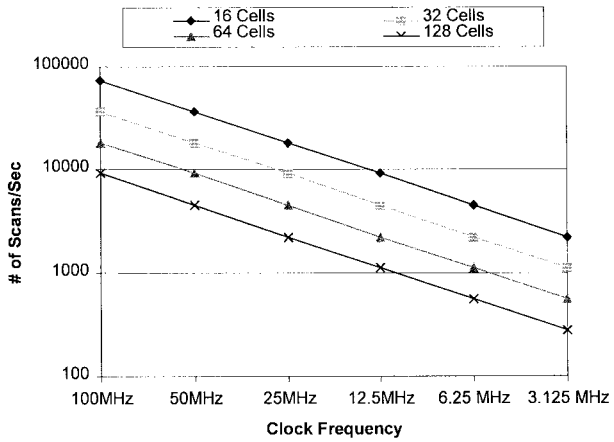


Fig. 4. Column testing frequency for multiple scanning clock frequencies.

D. Testing Issues

An important aspect of the fault scanner is the latency between scans of a specific column

$$\tau_{\text{scan}} = [7 * (\text{clk}) + 5 * (2' * \text{clk})] * N$$

where clk is the scanning clock period, I is the number of input bits to each LUT, N is the number of columns in the FPGA, and the constants 7 and 5 represent the number of steps of each length in the testing algorithm. The longer step size (i.e., $2I * \text{clk}$) represents steps in which the functionality of a LUT must be sequenced through. It should be noted that the scanning clock period may be a multiple of the system clock. Fig. 4 shows the number of scans per second for an FPGA with four-input LUT's as the clk period and number of columns varies, and the numbers represented in the figure are shown in Table I.

E. Fault Identification

An I/O pin is used to indicate that a fault has occurred. The input to this pin is the logic OR of the data on all of the FB's. If a fault is detected, the entire FPGA is reconfigured to some default state. If the fault was a SEU, then this will fix the fault and operation continues. If a fault in the same cell persists, then it is most likely a SSF. The only way to fix such a fault, without replacing the FPGA, is to avoid using that cell.

The first step in avoiding the use of the faulty cell is to identify which cell contains the fault. When a fault occurs FPGA functionality is stopped. Most FPGA's have a read-back function for debugging purposes. Read-back allows the internal state of the FPGA to be viewed externally when the FPGA is inactive. Using read-back, it can be determined which FB indicated a fault, and thus which row. The column of the fault is stored in the state machine that keeps track of the current column being tested. Once the row and column of the faulty cell have been determined the fault has been identified. The fault can then be handled by a number of fault tolerance schemes. A possible fault tolerance scheme is discussed in a later section.

TABLE I
COLUMN OPERATING MODES

Mode	Time
Functional	(Number of Columns – 1) * (2*Column copy time + Column testing time)
Testing	Column copy time + Column testing time

V. ALTERNATE IMPLEMENTATIONS

In addition to the testing scheme described above, there are multiple variations on this scheme which could be implemented.

A. Multiple Column Testing

It is possible to speed up the time it takes to scan the entire FPGA by having multiple FC's. This would allow scanning of multiple sections of the chip in parallel. The FPGA could be partitioned into sections with an equal number of columns in each section. Each section would have its own FC, and the CDATA and FB for each section would be independent. The CDATA buses must be segmented so that all sections can transfer cell data to their respective FC concurrently. The FB must be segmented so that faults can be detected independently in each section, and so that there is no contention on the buses. Since the control signals from the TC are distributed on a masked version of the global control signals, it is possible to implement this scheme using only a single TC.

This scheme has the obvious disadvantage of requiring that more of the FPGA resources to be dedicated to testing purposes. However, it will increase the speed of testing for faults by approximately the number of sections. The percentage overhead for varying number of TC's on multiple sizes of FPGA's can be seen in Table II.

B. Moving Free Column

One alternate implementation of fault scanning would be to have a moving FC that scanned across the FPGA following the column to be tested. This would require copying the function of each column to be tested no further than an adjacent column. After each column is tested it becomes the new FC. This change avoids the two steps in the testing scheme which write the original configuration back into the column which was being tested, and then the one extra step of having a hand-off to bring the tested column back on-line (Table III). This modification decreases the time to scan a chip to

$$\tau_{\text{scan}} = [5 * (\text{clk}) + 4 * (2' * \text{clk})] * \text{N}.$$

Another advantage of such a scheme is that all columns are functionally identical. In the original scheme the FC needed extra capability at its input and output to allow it to connect to the inputs and outputs of any other column. In this scheme, a single cell and interconnect model can be used across the entire chip.

This approach results in a variation in time between testing of each column. The original scheme simply swept from one side of the FPGA to the other, and then restarted at the beginning. Thus, the time between testing passes of any column is always the same. This modified scheme has to scan back and forth across the chip, so the time between passes of a given column alternates depending on the scanning direction (Fig. 5).

TABLE II
TESTING PHASES

Phase	Description
LUT	Compare LUT to configuration memory
LUT inverted	Invert LUT and compare to inverted configuration memory
FF	Compare FF to configuration memory
FF inverted	Invert FF value and compare to inverted configuration memory

TABLE III
NUMBER OF SCANS OF AN FPGA IN A SECOND

# of cells in Column	a) Scanning Clock Frequency					
	100MHz	50MHz	25MHz	12.5MHz	6.25 MHz	3.125 MHz
16 Cells	71839	35920	17960	8980	4490	2245
32 Cells	35920	17960	8980	4490	2245	1123
64 Cells	17960	8980	4490	2245	1123	561
128 Cells	8980	4490	2245	1123	561	281

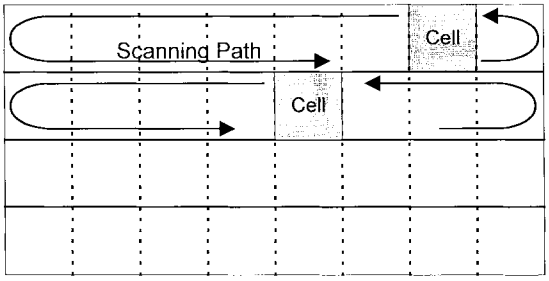


Fig. 5. Scanning routes for moving FC and TC.

This approach provides the advantage of identical columns, but at the expense of increased resources. Instead of expanding the input and output capabilities of a single column, the FC, as in the current scheme, the input and output capabilities of every column would have to be expanded to allow each column to function as its neighbor.

C. Moving Testing and Free Columns

Another option is to have both the FC and the TC move together. This approach removes the problem of having the control signals traveling across the chip. Using a moving TC means that local interconnect could be used to distribute control signals. There are other issues that arise, however. First, cell-to-cell interconnections must be increased by one column more than in the scheme where only the FC moved, so that signals can be passed to columns across the TC and FC. Second, the extra algorithm steps saved in switching to a moving FC are replaced by steps to move the TC. It is still not necessary to reimplement a column's function after it has been tested, but the column that was just tested now becomes host for the TC.

D. Multiple Testing Columns

Another possible implementation variation is to have multiple TC's with the above scheme (Table IV). That is, to have

TABLE IV
RESOURCE OVERHEAD FOR SCANNING SYSTEM VERSUS NUMBER OF TESTING COLUMNS USED FOR CONCURRENT SCANNING

# of cells / Column	% Overhead		
	1 Scanner	2 Scanners	3 Scanners
16 Cells	12.50%	25.00%	37.50%
32 Cells	6.25%	12.50%	18.75%
64 Cells	3.13%	6.25%	9.38%
128 Cells	1.56%	3.13%	4.69%

a similar scenario as describe in the multiple column testing case, but with a dedicated TC for each section. This would decrease the amount of time between scans for each column, as in the multiple column testing case, but would require an increase of two times the number of sections in resource usage for testing purposes. A major advantage of this approach over that of the multiple-column-testing case is that the control signal interconnect can be partitioned for each section. This would allow control signals to arrive slightly more quickly. This case has the disadvantages, however, of requiring an extra column for each new TC, and of requiring the segmentation of the control signal interconnect *a priori* in order to take advantage of the multiple TC's.

E. Point-To-Point Interconnect

Many FPGA's utilize point-to-point (or segmented) interconnect for routing (e.g., Xilinx XC4000). It may be possible to implement fault scanning on such FPGA's, but with significant modifications. Segmented interconnect consists of wires which make specific point-to-point connections. Each cell has multiple wires available each of which connects that cell to a specific other cell.

In the XC4000 parts these wires have specified lengths. There are wires to connect to cells one, two, four, and 16 cells away. Thus a connection can be made to a cell which is five cells away by routing it through an intermediate cell using the four-cell wire and then making use of the nearest-neighbor connection of that intermediate cell to route to the desired destination cell.

In order to implement fault scanning on an FPGA using segmented interconnects the ability to forward information to and from a column would be necessary. This would allow a FC to share the same input and output connections as the column it is mimicking. Any inputs or outputs to the column being tested would be routed the extra distance to the copy of that column. This forwarding of data would need to be taken into account during the place-and-route and simulation steps, as it means that the delay associated with a given connection can change over time.

Another necessary modification would be to the FC and TC. Instead of having a stationary FC and TC, the FC and TC would migrate across the FPGA. Having the FC and TC localized to the column being tested minimizes the amount of interconnect necessary. Minimizing interconnect is necessary because interconnect takes up the majority of the area in the FPGA and can, thus, be a limiting factor in FPGA design. Longer interconnect also increases the amount of time for testing and can limit the scanning speed of the FPGA. A migrating FC and TC scheme could make use of two-cell distance wires to forward information to and from the new copy of the column being tested, and the nearest-neighbor, or one-cell distance, wires to perform testing.

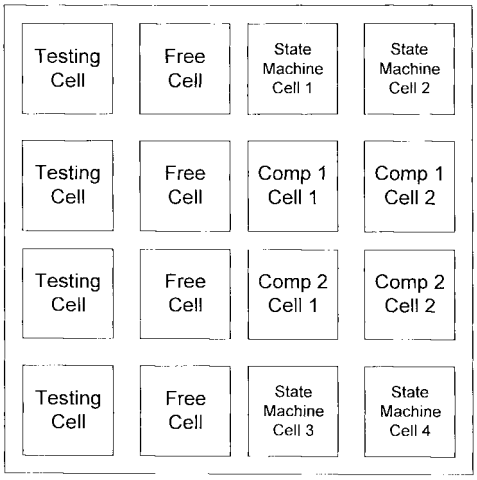


Fig. 6. Simulation cell functionality.

As a consequence of these issues, it seems unlikely that the fault-scanning approach could be implemented efficiently on a device without bus-based interconnect.

F. Fault Tolerance

It is possible to increase the capability of the fault scanning system to include some degree of fault tolerance as follows. If a fault is found in a cell, the cell's configuration is loaded into the FC cell on the same row as the faulty cell. If the fault is determined to be an SSF fault, then the FC cell can be switched to the inputs and outputs of the faulty cell and the faulty cell itself can simply be taken off-line. This method of fault tolerance can only accommodate a single fault in each row for each FC. An approach similar to this one is used during manufacturing test for the Altera Flex10K, though resources are switched through OTP fuses in manufacturing. This approach has the benefit of not impacting circuit performance, as signals are routed across the entire bus length regardless of the final destination column. Column replacement can only be used to avoid a single column for each available FC. Wider scale faults must be addressed using other techniques, such as those proposed by Lach *et al.* [29].

VI. EVALUATION

Simulation and testing was done using a Pentium Pro with the V-System simulator by ModelTech.

The FPGA simulation has been implemented in VHDL. A PERL script generates some of the VHDL files, allowing the simulation to handle any size of FPGA. This script takes as input the number of rows and columns in the FPGA to be simulated, and generates VHDL for an FPGA of the specified size.

In addition to the VHDL code, the simulation makes use of six files that specify control signal and input data. These files provide external input to the simulated FPGA.

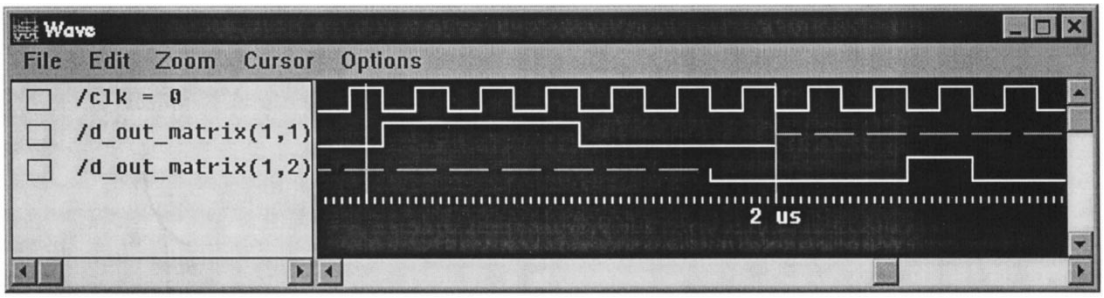


Fig. 7. Waveform of hand-off of functionality from column one to the free column (column two).

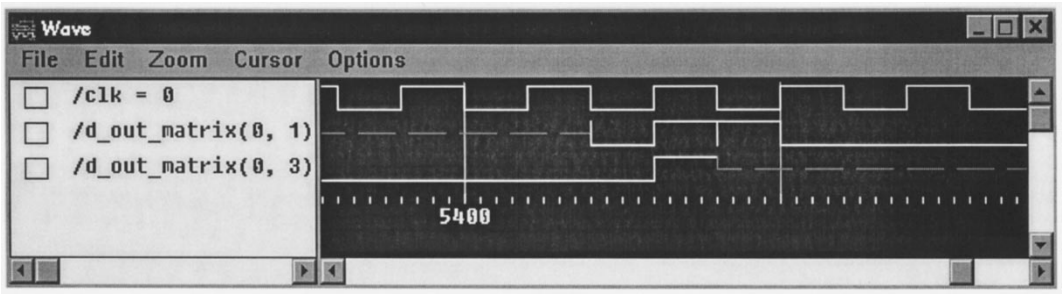


Fig. 8. A write taking place during a copy operation.

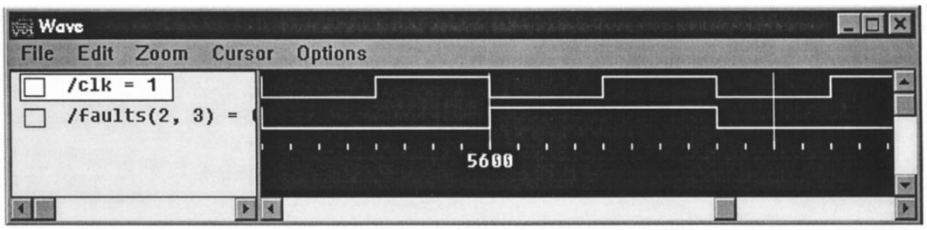


Fig. 9. A fault is detected in the LUT of cell (1, 3).

The FPGA simulated here has four rows and four columns. A 4×4 array is large enough to implement nontrivial functions in the two active columns and allow scanning to be tested, but is small enough that creating configuration data by hand is feasible.

The scanning clock rate for the simulated FPGA was 25 MHz, a reasonable rate for existing FPGA's. The scanning clock rate should be the highest possible multiple of the clock rate of the FPGA. This will allow scanning to proceed as fast as possible, while not requiring excessive amounts of resources to implement multiple clocks or synchronization of scanning with the operation of the rest of the FPGA.

Two different designs, each requiring four cells, were implemented simultaneously on the FPGA (Fig. 6). One design utilized only the combinational logic features of the cells to implement two comparators. Each comparator took as input two 3-bit words, A and B, and tested if A > B. Two comparators were implemented so that twice as many input vectors could be tested in a given period of time. Each comparator used two cells, spaced out over two adjacent columns.

The second design was a state machine that made use of both flip-flops and LUT's. The state machine itself was a simple design requiring only the remaining four cells. Three of the cells held state information, while the fourth implemented logic based on the current state. The state machine took as its inputs the system clock and a state machine reset signal. Unlike the comparators, the state machine implemented functionality across row boundaries in addition to column boundaries.

The simulation demonstrates that the on-line fault testing system is feasible. Avoiding glitches during the hand-off between the FC and column being tested, both when the control is given to the FC and when it is returned to the column being tested, is critical. Glitches are avoided by driving both columns with the same inputs, and enabling both columns' outputs to drive the same bus during the hand-off. Since the columns are configured identically, this technique results in the same output being driven onto the bus by both columns. The column to be taken off-line can then be disabled, and the other column takes over providing the function. This process is shown in Fig. 7, which displays the output signal for row one while control is being passed from the column being tested, column one, to the FC, column one. The dashed line in the figure represents tri-stating. Control is passed seamlessly, without any glitches on the output. The output shown here is from one of the comparator circuits.

A second critical system property is the assurance that writes to the flip-flops always occur properly. It is vital that no writes

be lost during the copying process. A mechanism of having writes take priority over the copy ensures that flip-flop values are copied as necessary, but that a stale value is never written. Fig. 8 shows a flip-flop write taking place during the copy operation, and shows that the correct value is written and produced by the flip-flop. There is a delay between writes and a change in the flip-flop output because the system is falling-edge triggered, while the flip-flops are rising-edge triggered.

A last critical system element is that faults be properly detected. Fig. 9 shows the discovery of an induced fault in the flip-flip element of a tested cell. The system accurately detects faults in both LUT's and flip-flops.

VII. Conclusion

The ability of reconfigurable systems to self-diagnose on-line is important to the viability of their use in many environments. Techniques for on-line fault identification and fault tolerance have been presented here. These techniques assume a bus-based FPGA architecture and a small state-machine that is assumed to be fault free. Such techniques provide the assurance of proper device functionality in a continuous manner with low hardware overhead. This capability allows faults to be identified and handled as quickly as possible, in the least intrusive manner possible. The multiple different fault detection techniques allow a tailoring of the fault monitor used for the system on which the monitor will be implemented.

A simulation has been created in order to prove the feasibility of such techniques. The simulation shows that fault detection can occur without affecting device functionality, and at a high enough rate to ensure an appreciably small amount of time between a fault's occurrence and its detection.

References

[1] D. P. Siewiorek and R. S. Swartz, *Reliable Computer Systems: Design and Evaluation.* Burlington, MA: Digital, 1992.

[2] T. J. O'Gorman, J. M. Ross, A. H. Taber, J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, and J. L. Walsh, "Field testing for cosmic ray soft errors in semiconductor memories," *IBM J. Res., Develop.,* vol. 40, pp. pp. 41–50, 1996.

[3] S. Trimberger, K. Doung, and B. Conn, "Architecture issues and solutions for a high-capacity FPGA," in *Proc. FPGA'97,* Monterey, CA, 1997.

[4] J. Rose, R. Francis, and P. Chow, "Architectures of field-programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE J. Solid-State Circuits,* pp. 1217–1225, 1990.

[5] Altera, *Data Book.* San Jose, CA: Altera, 1996.

[6] K. W. Bernhardt, "Advanced technologies for a command and data handling subsystem in a 'better, faster, cheaper' environment," in *Proc. Digital Avionics Systems Conf.,* Cambridge, MA, 1995.

[7] M. S. Abadir and H. Reghbati, "Functional testing of semiconductor random access memories," *Comput. Surv.,* vol. 15, pp. 175–198, 1983.

[8] V. D. Agrawal and S. C. Smith, *Test Generation for VLSI Chips.* Washington DC: IEEE Comput. Soc. Press, 1988, ch. 2.

[9] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Designs.* New York: Computer Science Press, 1990.

[10] J. P. Hayes, "On modifying logic networks to improve their diagnosability," *IEEE Trans. Comput.,* vol. C-23, pp. 56–62, Jan. 1974.

[11] K. N. Levitt, M. W. Green, and J. Goldberg, "A study of the data communication problems in self-repairable multiprocessors," in *Proc. Conf. AFIPS,* vol. 32. Washington, DC: Thompson, 1968, pp. 515–527.

[12] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self-test," *IEEE Design, Test of Comput., Mag.* vol. 10, 1993, pp. 69–77.

[13] C. Stroud, S. Konala, P. Chen, and M. Ambramovici, "Built-in self-test of logic blocks in FPGA's (finally, a free lunch: BIST without overhead!)," in *Proc. IEEE VLSI Test Symp.,* 1996.

[14] W. K. Huang and F. Lombardi, "An approach for testing programmable/configurable field programmable gate arrays," in *Proc. IEEE VLSI Test Symp.,* 1996.

[15] H. Michinishi *et al.,* "A test methodology for interconnect structures of LUT-based FPGA's," in *Proc. Asian Test Symp.,* 1996.

[16] T. Inoue *et al.,* "Universal test complexity of field-programmable gate arrays," in *Proc. Asian Test Symp.,* 1995.

[17] X. T. Chen, W. K. Huang, F. Lombardi, and X. Sun, "A row-based FPGA for single and multiple stuck-at fault detection," in *Proc. IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Syst.,* 1995.

[18] C. Jordan and W. P. Marnane, "Incoming inspection of FPGA's," unpublished, 1993.

[19] K. Kwiat, W. Debany, and S. Hariri, "Effects of technology mapping on fault-detection coverage in reprogrammable FPGA's," *Inst. Elect. Eng.: Comput., Digital Tech.,* 1995.

[20] G. A. Mojoli *et al.,* "KITE: A behavioral approach to fault-tolerance in FPGA-based systems," in *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems,* Boston, MA, 1996.

[21] R. Cuddapah and M. Corba, "Reconfigurable logic for fault tolerance," in *Field Programmable Logic and Applications*. Oxford, U.K.: Oxford University Press, 1995.

[22] F. Hanchek and S. Dutt, "Node-covering based defect and fault-tolerance methods for increased yield in FPGA's," in *Proc. 9th Int. Conf. VLSI Design,* 1995, pp. 225–229.

[23] N. J. Howard *et al.,* "The yield enhancement of field-programmable gate arrays," *IEEE Trans. VLSI Syst.,* vol. 2, pp. 115–123, 1994.

[24] K. Roy and S. Nag, "On routability for FPGA's under faulty conditions," *IEEE Trans. Comput.,* vol. 44, pp. 1296–1305, 1996.

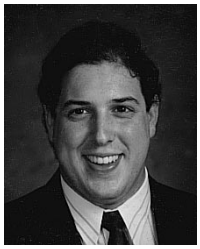
[25] J. L. Kelly and P. A. Ivey, "Defect tolerant SRAM based FPGA's," in *Proc. Int. Conf. Computer Design,* 1994.

[26] G. Swift and R. Katz, "An experimental survey of heavy ion induced dielectric rupture in actel field programmable gate arrays (FPGA's)," *IEEE Trans. Nucl. Sci.,* vol. 43, pp. 967–972, 1996.

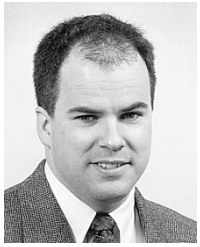
[27] K. A. LaBel *et al.,* "Single event effect proton and heavy ion test results for candidate spacecraft electronics," in *Proc. IEEE Radiation Effects Data Workshop,* 1994.

[28] L. A. Shombert and D. P. Siewiorek, "Using redundancy for concurrent testing and repairing of systolic arrays," presented at FTCS-17, Pittsburgh, PA, 1987.

[29] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE Trans. VLSI,* vol. 6, 1998.



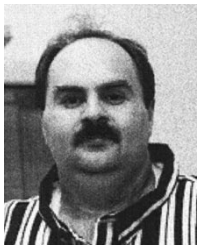
Nathan R. Shnidman received the B.Sc. and M.Eng. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (M.I.T.), Cambridge, in 1995 and 1996, respectively. He is currently pursuing the Ph.D. degree at MIT in the field of energy efficient radio frequency circuits and systems.



William H. Mangione-Smith (M'97) attended the University of Michigan, Ann Arbor, where he received the B.S.E. degree in electrical engineering in 1987 and the M.S.E. and Ph.D. degrees in computer science and engineering in 1992.

From 1991 to 1995, he was employed by Motorola Inc., where he participated in the design of the Envoy Personal Digital Assistant. Since 1995, he has been an Assistant Professor in the Electrical Engineering Department at the University of California, Los Angeles. His research interests include using dynamic circuits to implement configurable computing systems, low power processor and system design, multimedia and communications processing, and all techniques for leveraging instruction-level parallelism.

Dr. Mangione-Smith is a co-PI on the Mojave configurable computing program and served as the program chair for MICRO 26.



Miodrag Potkonjak received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1991.

In September 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been an Assistant Professor in the Computer Science Department, University of California, Los Angeles. His research interests include intellectual property protection techniques, system design, collaborative design, integration of computations and communications, and experimental algorithmics.

Dr. Potkonjak received the National Science Foundation (NSF) Career Award in 1998.