Fault Scanner for Reconfigurable Logic

Nathan R. Shnidman[1], William H. Mangione-Smith[1], and Miodrag Potkonjak[2]
Departments of Electrical Engineering[1] and Computer Science[2]
The University of California, Los Angeles
{naters, billms}@icsl.ucla.edu, miodrag@cs.ucla.edu

Abstract

We propose a technique for online built-in self-test of Field Programmable Gate Arrays (FPGAs). The goal of this system is to detect deviations from the intended functionality of an FPGA without using special-purpose hardware, hardware external to the device, and without interrupting system operation. A system that solves these problems would be useful for mission-critical applications with resource constraints. We present here a fault detection system which solves these problems through an online fault scanning methodology. Resources internal to the device are configured to test for faults. Testing scans across an FPGA, checking a section at a time. The viability and effectiveness of such a system is supported through simulation of the system on a model FPGA.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are the most common type of programmable logic devices today. FPGAs are made up of an array of programmable gates. The programmability of the FPGA resides in the ability to replace the function of each gate and to change the connectivity among gates.

Recent advances in FPGAs have precipitated their use in consumer applications, in addition to their more traditional use in logic emulation systems, prototyping, and low volume applications. Unfortunately, many of the trends that make newer FPGAs more appealing and affordable also make them less reliable. For example, smaller feature sizes, and the corresponding lower threshold voltages, make high density programmable devices more susceptible to gamma particle radiation. Also, larger die sizes make interference from such radiation much more likely [1].

Ironically, while hardware reuse is a primary reason for utilizing an FPGA, external hardware for fault testing and tolerance of FPGAs 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 writable memory of an FPGA will not be detected by such a system without adding a time consuming readback 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 reconfigurability. Such a fault monitor could run continuously while not impeding device functionality. This approach allows for rapid detection of both hard and soft faults.

0-8186-7913-1/97 $10.00 © 1997 IEEE

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 offline (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 impeding functionality.

1.1. FPGA architectures

FPGAs 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 is usually anywhere from 1 to 6 gates. Part of the programmability of the FPGA comes from the fact that the designer can change the actual type of gates implemented by each cell. In addition, the user also determines if the combinational logic section of the cell or the memory section of the cell is used. It is also sometimes possible to use both sections concurrently.

There are two basic models for the combinational logic portion of cells. One is called Island based, the other is called fine-grained [2]. The island based FPGA uses one or more look-up-tables (LUTs) per cell to provide the functionality of gates. These cells typically have four or more inputs. In contrast, a fine-grained FPGA's cells usually have only two inputs. The small number of inputs often allows logic in a fine-grained cell to be implemented with multiplexers [3].

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 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 [4]. The Flex10K uses static memory based LUTs, with a single flip-flop (Figure 1). In addition to the basic cell structure, the Flex10K groups eight cells, or Logic Elements (LEs), into what is called a Logic Array Block (LAB). These LABs provide point-to-point interconnect on a small scale, for fast local communication. The LABs are arrayed in a grid pattern, and are connected by chip-length buses (Figure 2). 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.

1.2. Motivational example

An online internal fault detection system would be of particular use in space-based applications. Limited resources such as volume, weight, and power make the use of FPGAs 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 much more operational interference from radiation and charged particles than terrestrial systems. The probability of faults in a space situation is much higher. As such, it is imperative that faults be located and addressed quickly.

Also, in space based systems, proper functionality is mission-critical. If an FPGA fails, it is vital to the mission that the fault be detected and handled as soon as possible. The system must be capable of autonomous and automatic fault detection and handling.

A fault scanning system would address most of these concerns. The fact that the fault scanner uses resources internal to the FPGA precludes an increase in the use of 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 the fault and continue to function.

1.3. Paper organization

This paper presents multiple possible internal online fault scanning monitors for FPGAs, and simulations showing the proof-of-concept for one implementation in particular. We will discuss the available design options and the simulation in Section 2. This section will also address the specific FPGA used for the simulation and how it compares to current FPGA models. We will also discuss previous work in the area, and its implications on the work presented here in Section 3. The basic fault scanning system is developed in Section 4. Alternative faults scanners will be presented in Section 5, 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 6. Some concluding remarks are given in Section 7.

2. Preliminaries

The assumptions used in implementing the fault scanning system will be discussed here. The types of faults addressed by this system are the Single-Stuck Fault (SSF) [5] and the Single-Event Upset (SEU) [5, 6]. This fault model is sufficient to verify LUTs and flip-flops. Faults in the interconnect and control paths are not considered in the current design of the system.

The focus of the current fault detection system on the LUTs and flip-flops is reasonable. This is true even though the majority of the FPGA area is taken up by interconnect. This is because the fault detection system is interested in detecting, and can only be reasonably 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 LUTs 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. A mechanism has been included, however, to help detect faults in the configuration elements and datapaths. The ability to readback the configuration data which has been loaded into the FPGA would help to catch such FPGA physical defects. This capability is important as the online fault scanning system does not deal with faults in the configuration elements or datapaths. A readback capability would provide a high level check of the FPGA every time a new configuration is loaded.

Additionally, the FPGA cells must be modified as described below. The global control signals must be maskable, so that control signals can target certain cells when necessary. Finally, the FPGA must be partially reconfigurable. Some FPGAs do not 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 online 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.

3. Related work

There has been much work in areas related to the fault system described here. This 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 [5, 7, 8], as well as in FPGAs in an offline manner [9, 10]. There has been much work done using BIST and even BISR to detect and handle fabrication faults in order to improve fabrication yields [11-14]. There has also been work done dealing specifically with FPGA's susceptibility to faults in space based situations [15, 16].

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

4. Approach

The discussion here is broken down into a system overview, presentation of the basic algorithm, and a discussion of the general FPGA architecture used in testing.

4.1. 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, and can move throughout the whole FPGA systematically testing for faults.

4.2. Basic algorithm

Built in fault testing for FPGAs 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 online testing algorithm reserves two columns of the FPGA for testing. One of these columns, the Testing Column (TC), contains the testing state machine. This state machine produces the control signals that implement testing. If the state machine is too large

for a column in an FPGA architecture, then the state machine must be run off-chip, with the control signals as chip inputs. At the moment, a second state machine, which keeps track of the column being tested, is modeled as being off-chip. For designs of 20 cells per column or larger, this 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 a buffer space during testing.

The online testing algorithm consists of three basic steps: copy, test, and move (Figure 5). 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 (Figure 3). This data on the CDATA bus is written to the FC, thus making a functional duplicate of the column to be tested. Since only a single LUT bit can be accessed at a time, both the configuration memory and the FC's LUTs are sequenced through all possible input vectors in parallel, by connecting their inputs to a counter.

After the configuration memory is copied, 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 simply 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. These values are written to the FC flip-flops. If there is a write to a flip-flop in the column to be tested, while it is being copied, the write always wins over the copy. This priority ensures that outdated values are not copied into the FC flip-flops. If a write to a flip-flop of a cell in the column to be tested occurs during the copying process, the new 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 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. The inputs to the column under test are connected to the output of the counter that is driving the configuration memory. The functions of 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 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 LUTs 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. It is important to 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.

Once the testing is completed, the move step begins. This step involves transferring the original functionality back into the column that was just tested. This phase begins by writing the non-inverted values in the configuration memories back into the LUTs. The inputs to the LUTs 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.

4.3. System architecture

The basic cell configuration can be seen in Figure 3. The main functionality of the cell consists of a four-input, one-output memory based LUT, and a single flip-flop. The LUT is simply a static memory with the four address bits used as the inputs to the LUT. The values in the LUT are stored during configuration of the FPGA. The cell has multiple possible functional 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.

In addition to functionality similar to that of a typical FPGA cell, the cell in Figure 3 has extra elements that allow for 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 possible 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. It is important to note that, during normal functionality, all writes to a cell's flip-flop write the same data, simultaneously, to the configuration flip-flop. This guarantees that the data in the configuration flip-flop is updated and valid.

The cells are arrayed in a grid pattern in the FPGA and are connected by device-length buses (Figure 4). Horizontal buses carry the cell inputs, which are 4 bits 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. A bus also exists for the output of each cell in that 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 3 and column 2 are stored in the configuration memory of cell (2,3). The switches have the ability to write their state value to the CDATA bus (see below). 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. Any discrepancy between the two values signals a fault. Additionally, this 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, or currently stored, in the LUT and flip-flip. 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 online 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 the global nature of these configuration 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. Thus, configuration data would have to be input sequentially by cell, in each row, instead of just configuring all cells in parallel. However, since 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). The FB is similar to the CDATA bus in that there is one such bus per row, 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 use of the FB in fault notification will be discussed later.

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 lookup tables in the cells. The simulated FPGA, however, does not make use of grouped cells (LABs) and dedicated memory blocks as does the Flex10K.

4.4. Testing issues

An important aspect of the fault scanner is that the time between subsequent scans of any cell is relatively low, this reducing operating time in a faulty mode. The time between consecutive scans of a given cell is:

$$\tau_{scan} = \left[7 * (clk) + 5 * \left(2^{I} * clk\right)\right] * N$$

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 seven and five represent the number of steps of each length in the testing algorithm. The longer step size (i.e. $2^{I} * clk$) 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. Figure 7 shows the number of scans per second for an FPGA with 4-input LUTs as the clk period and number of columns varies, and the numbers represented in the figure are shown in Table 1.

# of cells in Column	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

Table 1: Number of Scans of an FPGA in a second

4.5. 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 FBs. 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 FPGAs have a readback function for debugging purposes. Readback allows the internal state of the FPGA to be viewed externally when the FPGA is inactive. Using readback it can be determined which FB indicated a fault. That FB corresponds to a specific row. The state stored in the state machine that keeps track of the current column being tested denotes in which column the fault occurred. 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.

5. Alternate implementations

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

5.1. Multiple column testing

It is possible to speed up the time it takes to scan the entire FPGA by having multiple FCs. 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. If the sections do not contain exactly the same number of columns, however, the current column to be tested must be stored separately for each section width.

This scheme has the obvious disadvantage of requiring 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 TCs on multiple sizes of FPGAs can be seen in Table 2.

# of cells in 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%

Table 2 : Resource overhead for scanning system vs. number of testing columns used for concurrent scanning

5.2. 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 functional 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 testing, and then the one extra step of having a handoff to bring the tested column back online. This modification decreases the time to scan a chip to:

$$\tau_{scan} = \lfloor 5 * (clk) + 4 * (2^{l} * clk) \rfloor * N$$

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 to create the entire chip, which makes simplifies the design.

There are some disadvantages to this scheme. The first of these is that, while data interconnect is point-to-point, control interconnect is still global. The TC does not move, and as such, the control signals must travel across the chip. This long interconnect could limit the scanning speed of the FPGA. This issue will be addressed in the next section.

This approach also 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 (Figure 7).

A moving FC scheme could be implemented on a bus-based FPGA. This approach would provide the advantage of identical columns, but at the expense of increased resources. Instead of expanding the input and output capabilities of a single column, 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.

5.3. 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 travelling 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 re-implement a column's function after it has been tested, but the column that was just tested now becomes host for the TC.

5.4. Multiple testing columns

Another possible implementation variation is to have multiple TCs with the above scheme. That is, to have 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 TCs.

5.5. Point-to-point interconnect

Many FPGAs utilize point-to-point (or segmented) interconnect for routing (e.g. Xilinx 4K). It is possible to implement fault scanning on such FPGAs, but with some modifications. Interconnect would have to be added that allowed the transfer of configuration data to a FC without interrupting the normal function of a cell. The cell-to-cell interconnections would also have to be lengthened to allow a FC to share the same input and output connections as the column it is mimicking. The last major modification is to the concept of the FC itself. Instead of having a stationary FC, the FC would migrate across the FPGA. Having the FC localized to the column being tested minimizes the amount of interconnect necessary. Interconnect increases for testing can limit the scanning speed of the FPGA, so it is vital that these increases be minimized.

5.6. Fault tolerance

It is possible to increase the capability of the fault scanning system to include 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 offline. 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.

6. Evaluation

Simulation and testing was originally done using the Cadence toolset (HDL desktop, Leapfrog VHDL simulator, and Waveview wave display) on a Sparc20. Later, simulation was moved to a Pentium Pro making use of 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 be independent of the size of the 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.

The FPGA simulated here has 4 rows and 4 columns. A 4x4 array is large enough to implement non-trivial 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 FPGAs. 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 (Figure 8). One design utilized only the combinational logic features of the cells to implemented two comparators. Each comparator took as input two three-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 required two cells, spaced out over two adjacent columns.

The second design was a state machine that made use of both flip-flops and LUTs. 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 online 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 Figure 9, which displays the output signal for row 1 while control is being passed from the column being tested, column 2, to the FC, column 1. 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. Figure 10 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. Figure 11 shows the discovery of an induced fault in the flip-flip element of a tested cell. The system accurately detects faults in both LUTs and flip-flops.

7. Conclusion

The ability of reconfigurable systems to self-diagnose and even self-repair online is important to the viability of their use in many environments. Techniques for online fault identification, and fault tolerance have been presented here. Such techniques provide the assurance of proper device functionality in a continuous manner with little 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 to 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 Press, 1992.

[2] S. Trimberger, K. Duong, and B. Conn, "Architecture Issues and Solutions for a High-Capacity FPGA," presented at FPGA97, Monterey, CA, 1997.

[3] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuit*, vol. 25, pp. 1217-1225, 1990.

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

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

[6] J. P. Hayes, "On Modifying Logic Networks to Improve Their Diagnosability," *IEEE Transactions on Computers*, vol. 23, pp. 56-62, 1974.

[7] K. N. Levitt, M. W. Green, and J. Goldberg, "A Study of the Data Communication Problems in Self-Repairable Multiprocessors," presented at AFIPS, Washington, D. C., 1968.

[8] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-in Self-Test," *IEEE Design and Test of Computers*, vol. 10, pp. 69-77, 1993.

[9] C. Stroud, S. Konala, P. Chen, and M. Ambramovici, "Built-in Self-Test of Logic Blocks in FPGAs (Finally, a Free Lunch: BIST Without Overhead!)," presented at Proceedings of the 14th IEEE VLSI Test Symposium, 1996.

[10] W. K. Huang and F. Lombardi, "An Approach for Testing Programmable/Configurable Field Programmable Gate Arrays," presented at the 14th IEEE VLSI Test Symposium, 1996.

[11] F. Hanchek and S. Dutt, "Node-Covering Based Defect and Fault-Tolerance methods for Increased Yield in FPGAs," presented at the Ninth International Conference on VLSI Design, 1995.

[12] N. J. Howard, A. M. Tyrrell, and N. M. Allinson, "The Yield Enhancement of Field-Programmable Gate Arrays," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 115-123, 1994.

[13] K. Roy and S. Nag, "On Routability for FPGAs Under Faulty Conditions," *IEEE Transactions on Computers*, vol. 44, pp. 1296-1305, 1996.

[14] J. L. Kelly and P. A. Ivey, "Defect tolerant SRAM based FPGAs," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 479-482, 1994.

[15] G. Swift and R. Katz, "An Experimental Survey of Heavy Ion Induced Dielectric Rupture in Actel Field Programmable Gate Arrays (FPGAs)," *IEEE Transactions on Nuclear Science*, vol. 43, pp. 967-972, 1996.

[16] K. A. LaBel, A. K. Moran, D. K. Hawkins, J. A. Cooley, and e. al., "Single Event Effect Proton and Heavy Ion Test Results for Candidate Spacecraft Electronics," *IEEE Radiation Effects Data Workshop*, pp. 64-71, 1994.

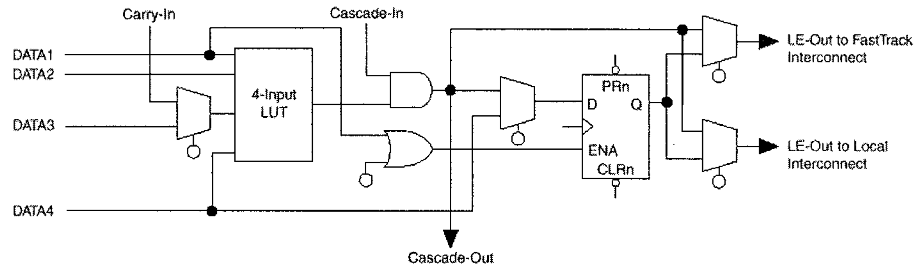

Figure 1: Flex10K cell [4]

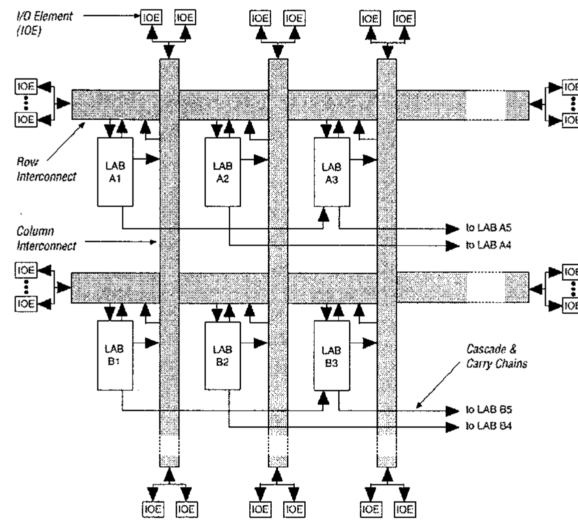

Figure 2: Flex10k Array [4]

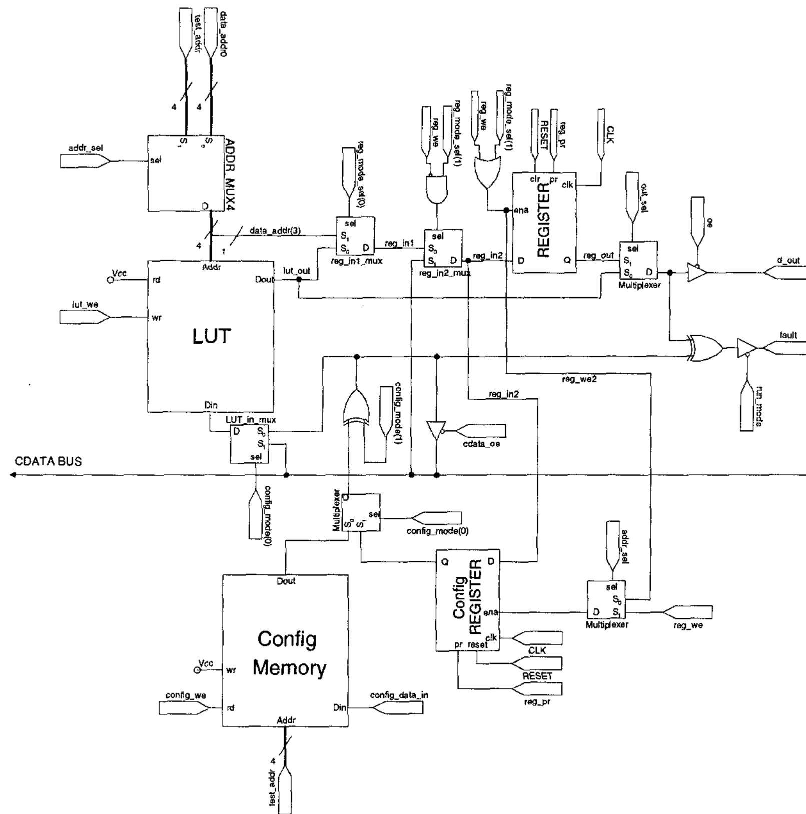

Figure 3:Cell design for the simulated FPGA (includes configuration elements and special logic to allow scanning)

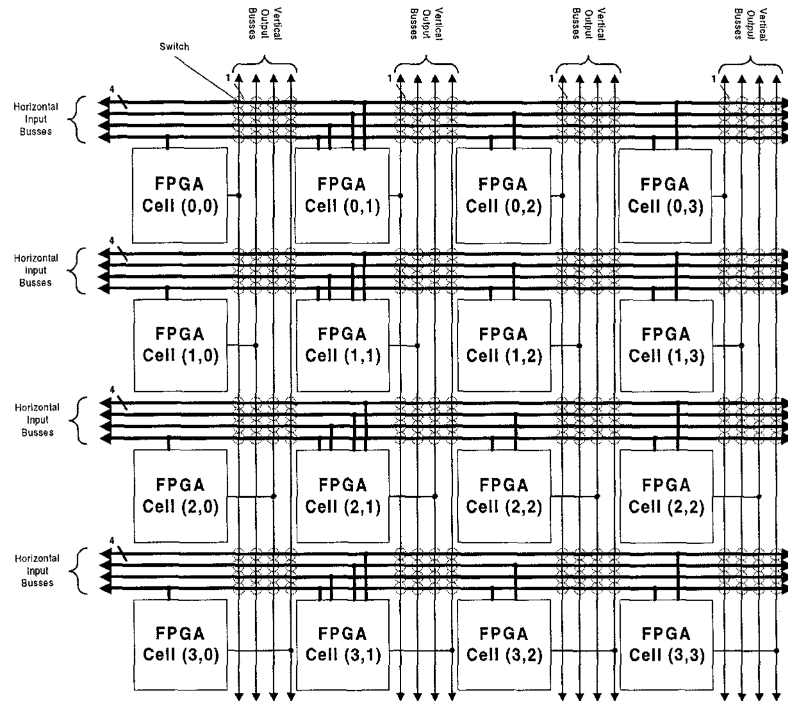

Figure 4: FPGA Cell Array with Interconnect

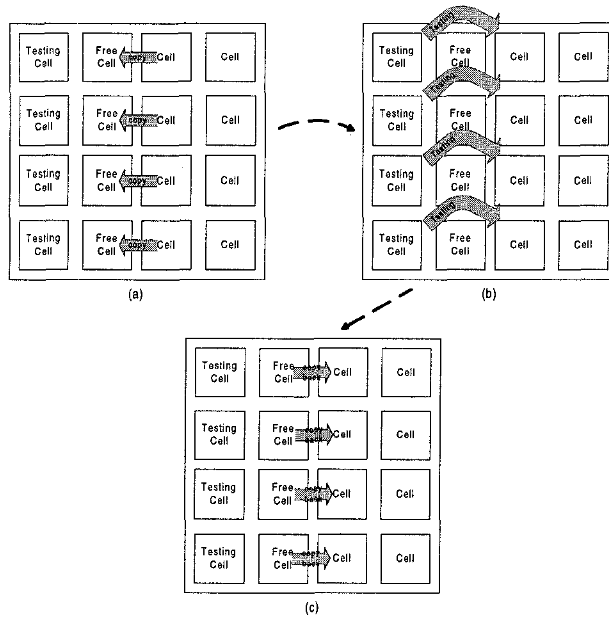

Figure 5: Basic Algorithm: move functionality of column to be tested to free column (a), test the column (b), copy the functionality back (c)

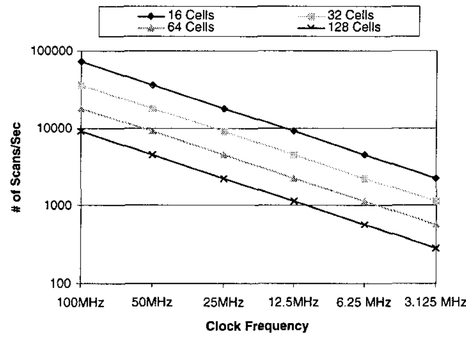

Figure 6: Number of times an FPGA of a given size can be scanned in one second, for multiple scanning clock frequencies

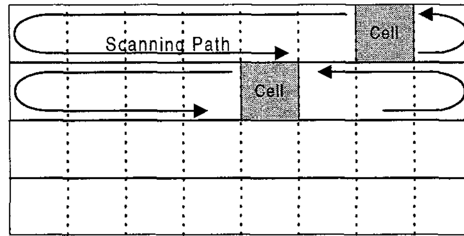

Figure 7: Scanning routes for moving FC and TC

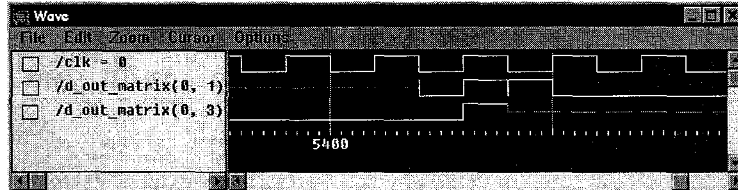

Figure 8: A write taking place during a copy operation

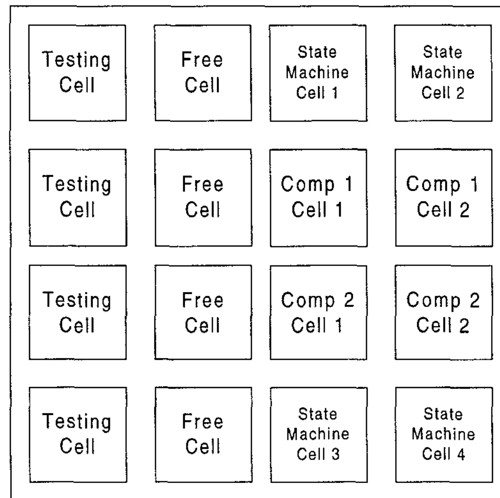

Figure 9: Simulation Cell Functionality

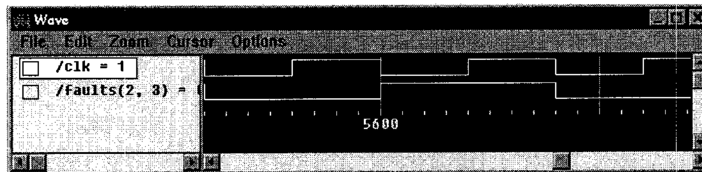

Figure 10: A fault is detected in the LUT of cell (1,3)

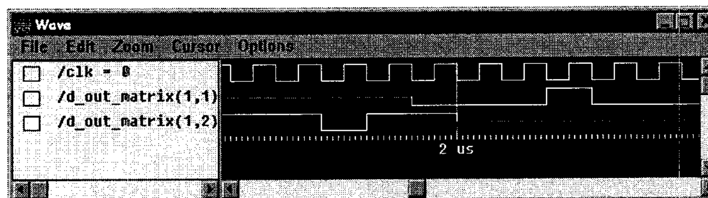

Figure 11: Waveform of hand-off of functionality from column 2 to the Free Column (column 1)