

# Enhanced FPGA Reliability Through Efficient Run-Time Fault Reconfiguration

John Lach, *Member, IEEE*, William H. Mangione-Smith, *Member, IEEE*, and Miodrag Potkonjak, *Member, IEEE*

**Abstract**—The expanded use of field programmable gate arrays (FPGA) in remote, long life, and system-critical applications requires the development and implementation of effective, efficient FPGA fault-tolerance techniques. FPGA have inherent redundancy and in-the-field reconfiguration capabilities, thus providing alternatives to standard integrated circuit redundancy-based fault-recovery techniques. Runtime reliability can be enhanced by using such unique features. Recovery from permanent logic and interconnect faults without runtime computer-aided design (CAD) support can be efficiently performed with the use of fine-grained physical design partitioning. Faults are localized to small partitioned blocks that have fixed interfaces to the surrounding portions of the design, and the affected blocks are reconfigured with previously generated, functionally equivalent block instances that do not use the faulty resources. This technique minimizes the post-fault-detection system downtime, while requiring little area overhead. Only the finely located faulty portions of the FPGA are removed from use. In addition, the end user need not have access to CAD tools, making the algorithm completely transparent to system users. This approach has been efficiently implemented on a diverse set of FPGA architectures. The algorithm's flexibility is also apparent from the variable emphases that can be placed on system reliability, area overhead, timing overhead, design effort, and system memory. Given user-defined emphases, the algorithm can be modified to specific application requirements. Experiments using random  $s$ -independent and  $s$ -correlated fault models reveal that the approach enhances system reliability, while minimizing area and timing overhead.

**Index Terms**—Fault-tolerance, field programmable gate array (FPGA), online.

## NOMENCLATURE

### Acronyms<sup>1</sup>

AFTB	atomic fault-tolerant block
CAD	computer-aided design
CLB	configurable logic block
CSLA	context switching logic array
CSLC	context switching logic cell
CSRC	context switching reconfigurable computing
FPGA	field programmable gate array

Manuscript received November 1, 1999. This work was supported by the U.S. Air Force Research Laboratory under Contract F30602-96-C-0350 and Subcontract QS5200 from Sanders, a Lockheed Martin Company.

J. Lach is with the Electrical Engineering Department, University of Virginia, Charlottesville, VA 22904 USA (e-mail: JLach@virginia.edu).

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

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

Publisher Item Identifier S 0018-9529(00)11754-3.

<sup>1</sup>The singular and plural of an acronym are always spelled the same.

LAB	logic array block
LE	logic element
LUT	lookup table
STAR	self-testing area.

### Notation

$\parallel$	OR
$\&$	AND
$p$	probability of proper functionality (reliability)
$\mu$	cluster variability factor.

## I. INTRODUCTION

**R**ELIABILITY is crucial for remote and long-life systems, and quick recovery is necessary for critical applications that do incur faults. A substantial amount of research has been committed to increase reliability at various design levels, from the microarchitecture [1] to the system-level [2]. CAD synthesis and hardware design techniques have been proposed that focus on enhancing reliability [3], [4].

FPGA are commonly incorporated into systems requiring high reliability. Therefore, a low overhead fault recovery algorithm has been developed for FPGA that can recover from faults at runtime with minimal system downtime and no end-user CAD tool requirements. The approach takes advantage of the flexible and inherently redundant nature of FPGA. Assuming detection,<sup>2</sup> localization, and diagnosis<sup>3</sup> of a fault, a configuration of the design can be loaded that does not utilize the faulty resource(s). The alternate configurations are previously generated by the CAD tools at design-time and are available in memory. The proper configuration is then activated based on the location of the faults. The previously prepared configuration need only be applied to the device, thereby not requiring substantial system downtime or the end user to have access to CAD tools. Therefore, the algorithm, and even the very existence of an FPGA in the system, remains transparent to the user. A previous algorithm achieved such runtime fault recovery exclusively for logic faults [11], but the algorithm has been expanded to include interconnect faults and has been applied to a variety of FPGA architectures.

### A. Approach

This approach partitions the physical design into a set of  $s$ -independent blocks (tiles). Each tile is composed of a

<sup>2</sup>Many FPGA testing techniques are currently available for both logic and interconnect [5]–[10].

<sup>3</sup>Any fault that occurs can be considered permanent (the resource will no longer be used). Therefore, diagnosis is not entirely necessary, but it may be helpful in identifying nonpermanent faults that can be corrected.

- set of physical resources (*viz.*, logic blocks and interconnect),
- an interface specification which denotes the connectivity to neighboring tiles,
- a netlist.

Logic and local interconnect reliability are achieved by providing multiple configurations of each tile, each of which does not use certain resources within the tile. Furthermore, by using immutable tile interfaces, the effects of swapping a tile configuration do not propagate to other tiles, thereby making each tile *s*-independent and reducing the storage overhead.<sup>4</sup> Any paths that cross tile boundaries can be made reliable by reserving other inter-tile interconnect to be used as spares (see Section I-B for examples).

This approach has three main benefits compared to redundancy-based fault recovery techniques:

- very low overhead,
- the option for runtime management and transparency,
- flexibility.

The area overhead, measured in physical resources on the FPGA (logic blocks, I/O blocks, and interconnect), required to implement this fine-grained approach is extremely low compared to redundancy. This is due primarily to the inherent redundancy in FPGA, as opposed to the introduction of redundant elements into fixed designs that is required for reliability and yield enhancement [12]–[14]. Timing overhead, measured in the circuit performance, is also kept low. Based on the system constraints, the emphasis placed on area and timing results in different tiling configurations.

Runtime management and transparency can be very valuable features of a system, particularly for mission-critical applications. This fault-recovery approach handles runtime problems on-line, minimizing the amount of system downtime and eliminating the need for outside intervention. The end user is not required to have the CAD tools that are otherwise necessary to alter the layout of the circuit, as the configurations are all generated at design time. Therefore, the technique is completely transparent to the user. Current nontransparent techniques do exist that provide efficient runtime fault-tolerance for FPGA. The incremental rerouting approach [15] reconfigures designs with minimal perturbation at runtime (resulting in lower timing overhead) and has a short re-layout time, but the end user is required to have the layout tool on the system. The same is true for the roving STAR technique [16], which efficiently provides runtime fault testing, diagnosis, and bypass. Although effective, these techniques cannot be used if transparency is a strict system requirement.

The flexibility that this approach provides allows for application-specific solutions. The degree of fault recovery can be adjusted to meet timing and resource constraints or estimated logic block and interconnect reliability.

<sup>4</sup>Tile *s*-independence requires the system to generate and store individual tile information and its corresponding instances. However, no inter-tile information need be generated or stored, thus reducing CAD tool effort and storage requirements.

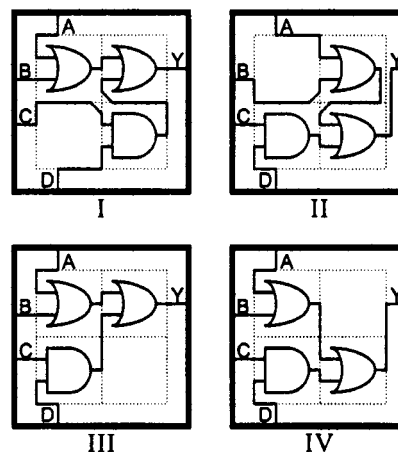


Fig. 1. Logic fault recovery.

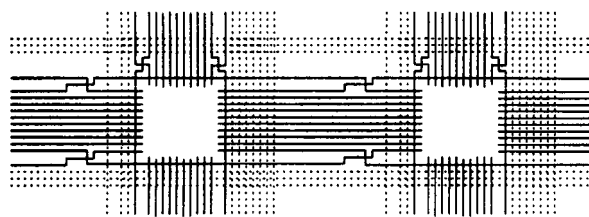


Fig. 2. Inter-tile interconnect fault recovery.

### B. Example

Consider the Boolean function

$$Y = (A||B) || (C&D),$$

which might be implemented in a tile containing 4 logic blocks; see Fig. 1. This partitioning contains 1 spare logic block, which is available if a fault is detected in 1 of the occupied logic blocks. Upon detecting such a fault, an alternate configuration of the tile is activated which does not rely on the faulty logic block. Each implementation is interchangeable with the original, because the interface between the tile and the surrounding areas of the design is fixed and the individual configurations implement the same function.

Recovering from local interconnect faults are handled in much the same manner. Interconnect can be set aside as unused in each tile configuration,<sup>5</sup> and when a fault disables an interconnect line, an instance not using that line can be activated.

Recovering from faults to global and overlapped segmented interconnect requires a different approach, because much of that interconnect crosses tile boundaries, thus eliminating the *s*-independent nature of each tile. Ignoring tile boundaries, interconnect can be set aside that acts as backup for used global and overlapped segmented interconnect. However, the backup interconnect must be able to fulfill the connections of the failed interconnect without requiring an alteration to the affected tiles. Fig. 2 shows how a global line (dotted) can act as a backup for several segmented interconnect lines (solid) in the Xilinx XC4000 family. When a segmented line sustains a fault,

<sup>5</sup>Throughout the configuration generation for logic reliability, most local interconnect is unused in at least one instance, making it unnecessary to generate many additional instances for local interconnect reliability.

the backup global line can be activated. This requires only the programming of the proper connections and does not affect the logic in any way.

## II. TILING

One approach to FPGA online fault recovery is to generate different instances of the entire design, leaving some resources unused in each instance. When a fault is detected and localized, the entire design is reconfigured with the design instance that does not use the faulty resources. However, tiling achieves increased reliability while reducing the design effort and the memory needed to create and store the set of instances due to the fine granularity that tiling provides. For example, consider a design that maps into a  $6 \times 6$  logic block array with 4 unused logic blocks. Let one configuration of the complete  $6 \times 6$  design requires  $X$  units of effort and memory. If the 4 unused blocks were moved as a group to cover each logic block once, a total of 9 configurations would need to be created, or  $9X$  units of effort and memory. However, multiple  $s$ -independent faults can occur, and a configuration that covers them might not be available. Another approach would be to generate configurations for any 4 combinations of faults, but that would require

$$\binom{36}{4} \cdot X = 58905X$$

units of effort and memory.

Tiling can divide the design into 4  $s$ -independent  $3 \times 3$  tiles, each with 1 unused logic block. Each tile would be able to recover from any single logic block fault and up to 4 in the entire design,<sup>6</sup> and each of the 4 tiles would require 9 instances. However, since each tile ( $X/4$  units of effort and storage) is  $s$ -independent, only  $9X$  units of effort and storage are required. Therefore, compared to the technique that moves the unused resources as a cluster, fine-grained tiling provides substantially higher reliability with the same effort and storage requirements.

The key concepts for implementing the new approach are tiles and AFTB. Fixing the interfaces between the tiles, creates multiple partial configurations (AFTB) that satisfy the functional specification for a given tile, independently from the remainder of the design. Fault-tolerance is achieved by introducing spare resources into each AFTB so that, once a fault in a particular resource is detected and localized, a configuration of the tile's functionality that does not use the faulty resource can be activated. The locked tile interfaces ensure that configuring any AFTB to its tile does not affect the rest of the design, with the exception of a potential timing impact.

Starting from an original nonfault-tolerant base design, recursively partition the design into tiles and AFTB, looking for feasible solutions that meet the specific user-defined area, timing, and reliability requirements. The approach is detailed in the following pseudo-code:

### Algorithm

1. while (!(complete || design possibilities exhausted)) {
2.     create initial non\_ft\_design;

<sup>6</sup>If the desired system reliability requires that multiple faults be tolerated within each tile, additional unused resources can be inserted into each tile. This slightly increases area overhead, but can greatly enhance reliability.

3.     extract timing and area information;
4.     calculate design reliability;
5.     while (!(complete || tiling possibilities exhausted)) {
6.         partition design into tiles;
7.         if (!meet area criteria) break;
8.         while (!(complete||AFTB possibilities exhausted))
9.             partition tiles into AFTB;
10.             calculate AFTB reliability;
11.             if (!meet reliability criteria) break;
12.             for ( $j = 1; j \leq \#$  of tiles;  $j++$ ) {
13.                 for ( $i = 1; i \leq \#$  of AFTB;  $i++$ ) {
14.                     create ft\_design ( $i, j$ );
15.                     if (!(success && meet timing criteria))
- break;
16.                 } } } } }
- End\_Algorithm

The base design is created in line 2, and the relevant design characteristics are recorded for future overhead and reliability enhancement calculations.

The fault-tolerance portion of the algorithm begins in line 5, which dictates that the loop terminates upon the creation of a fault-tolerant design that meets all of the user specifications (resource and timing overhead, reliability, and memory). If all of the tile partitioning possibilities are exhausted, the loop terminates, returning the algorithm to line 1 and the creation of a different base physical design.

Line 6 partitions the design into tiles based on the amount of interconnect across the tile interface (should be minimal), tile logic density (must be enough unused resources and malleability), macros (should be kept intact), and tile size (desired level of granularity). If the original tiling efforts do not meet the user area specifications, the design must be re-tiled.

AFTB are created beginning in line 8 with a loop that terminates upon the successful creation of a set of AFTB that meet all user specifications or upon the exhaustion of all AFTB partitioning possibilities. If the latter occurs, the design must be re-tiled beginning at line 5. The number of AFTB, partitioned in line 9, for a given tile depends on the desired level of reliability, the amount of free resources in the tile, and the malleability and density of the logic. Reliability is checked in lines 10–11, returning to line 5 if the requirements are not met. Lines 14–15 are repeated for each AFTB in each tile. If at any point the requirements cannot be met, the Algorithm returns to the AFTB partitioning loop at line 8.

Once all of the AFTB are stored in memory, the design is ready for run-time implementation. Upon detection and localization of a fault, the proper AFTB is located in memory and configured to the device, allowing a return to usual operation with minimal system downtime and no CAD tool requirements.

## III. FPGA ARCHITECTURES

Implementation of the Algorithm varies depending on the target FPGA architecture. Sections III-A–III-C describe the implementation on three architectures: Sanders' CSRC technology [17], Xilinx's XC4000 family [18], and Altera's Flex 10k series [19].

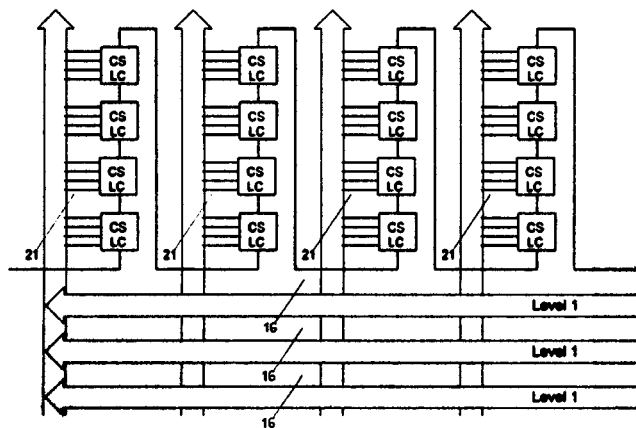


Fig. 3. CSLA and level-1 routing.

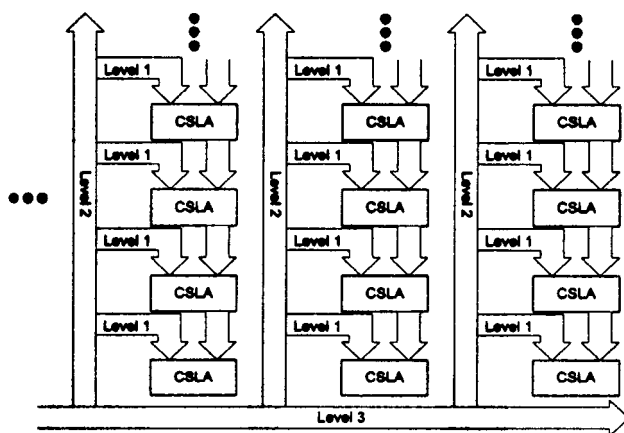


Fig. 4. Data pipes and level-3 routing.

### A. Sanders CSRC

The CSRC device is based on new architectural techniques that exploit dynamic reconfiguration via context switching. Each context is identical and is composed of logic and interconnect arranged in a hierarchical structure. At the lowest level of logic is the CSLC. The 16 CSLC comprise 1 CSLA, as shown in Fig. 3. Each logic cell has a carry-in and carry-out. The 16 CSLC are sectioned into 4 groups that are connected and driven by Level-1 routing. The next level of logic and interconnect is the 16-bit data pipe which is composed of CSLA and connected by Level-2 routing as shown in Fig. 4. Each context is composed of a set of data pipes that are connected by Level-3 routing, completing the highest level of logic and routing.

Developing a fault recovery approach for the CSRC device can be done in two ways:

- 1) Looking at each context as an  $s$ -independent entity,
- 2) Allowing functionality belonging to one context to be transferred to another in the face of a fault.

Analysis reveals that #2 is less desirable. Moving a section of a pipeline, for example, to another context would require switching to the other context in mid-stream before switching back again to complete the context's original function. The CSRC technology can perform such a task, but the performance

cost would be great. For example, leaving an entire context free to serve as a spare to be programmed and activated for a fault to a portion of an active context is undesirable. In general, such an approach would require a tremendous amount of overhead ( $X$  spare contexts for every  $Y$  active contexts<sup>7</sup>) and only be able to recover from  $X$  faults throughout the life of the system. Fault recovery in FPGA can be implemented efficiently because of the inherent redundancy (just as contexts are redundant), but the larger the redundant block, the higher the resource overhead and the smaller number of tolerable faults. Allocating or reserving spare contexts in the CSRC device is analogous to simply adding redundant FPGA in a non-CS system.

Looking at each context as an  $s$ -independent entity, a much finer grained approach, helps to alleviate these problems. For example, looking at smaller blocks of redundancy (CSLA or even CSLC) creates the opportunity for lower resource overhead and a smaller performance impact. CAD tools rarely map logic to maximum density, leaving many CSLA and CSLC unused. Using these as redundant blocks eliminates the effective resource overhead. Additional unused resources can, and often should, be added to raise the number of tolerable faults, thus creating resource overhead. Both the naturally unused and additional redundant blocks must also be distributed for easier fault recovery and a smaller performance impact, but the overhead is still appreciably reduced from a larger redundant block approach.

This approach also raises the number of recoverable faults and creates a more efficient and realistic fault model. Very rarely would a fault occur that destroys a large portion of the chip. If such a situation arose, it would be unlikely that enough of the chip would remain functional, thus rendering any on-chip fault recovery algorithm useless. Most faults that would occur are single faults that affect a small segment of memory (e.g., LUT), logic (e.g., multiplexor or flip-flop), or interconnect at any level. Such a fault model dictates the use of smaller redundant blocks, because 1 faulty wire or LUT should not render an entire context faulty.

Breaking the CSRC device contexts into  $s$ -independent fault recovery blocks (tiling) can be done in several ways. Selecting the most efficient can be application dependent. The pipeline nature of the interconnect makes it difficult to have a single CSLC be redundant for the others in its array (see Fig. 3). If a logic cell in the middle of a 4-cell pipe portion should fail, it might not be possible for the CAD tool to route the proper signals to reach the redundant cell. Therefore, it becomes necessary to look at a slightly larger block, *viz.*, the 4-cell pipe portion. Each portion has the same connectivity, making it possible for 1 pipe portion to be redundant for any other in the array. Thus, each array has 1 4-cell pipe portion that is redundant for the other 3, and 3 other configurations are generated that would be instantiated upon a failure to 1 of the active pipe portions.

Almost all types of failures can be tolerated at this level. Faults to the cells, the cell pins, and the wires leading from the Level-1 routing to the cell pins can all be tolerated, as each can be attributed to a specific pipe portion. Simply switching to an

<sup>7</sup>Sanders' CSRC device has 4 contexts. With 3 active contexts and 1 spare, resource overhead is 33% while being capable of tolerating only 1 fault.

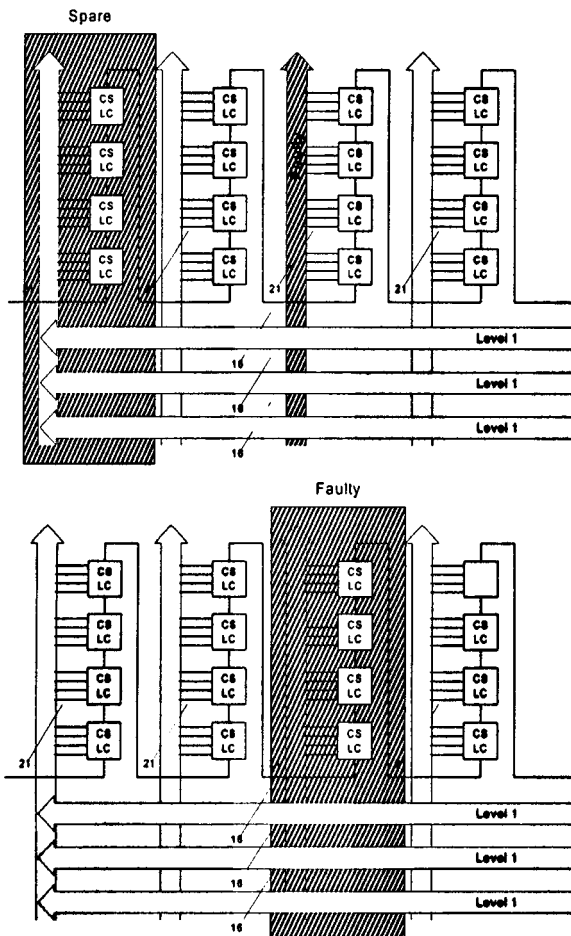


Fig. 5. Original and recovery configuration after an internal CSLA fault.

array configuration not utilizing the pipe portion containing the faulty hardware recovers from the fault. Fig. 5 shows how a CSLA may be reconfigured if a fault were to occur in the routing from Level-1 to the logic cell pin wires in pipe portion three (or anywhere within the third pipe portion).

One problem with this tiling approach concerns inter-pipe interconnect. This interconnect includes the Level-1 routing and the carry lines, as such lines cannot be attributed to a single pipe portion. The connections among Level-1 routing are plentiful and flexible enough that the CAD tools can find acceptable routing if a Level-1 line should fail, but configurations must be ready at runtime that do not necessitate in-the-field CAD tool use. Therefore, configurations must be generated at design-time that do not use each Level-1 line in at least one configuration, just as each pipe-portion is unused in at least 1 configuration.

Recovering from faults to the carry line is more difficult, because such a fault potentially renders 2 pipe portions inoperative. The backend CAD tool deals with the carry lines when there is a break in the pipe (i.e., a middle pipe portion is faulty and, therefore, unused) during its design-time configuration generation. However, if the carry-out of 1 pipe portion and the carry-in of another are faulty, an inter-pipe portion problem arises. The CAD tool might be able to implement the array's functionality without using the carry line (*viz.*, each carry line

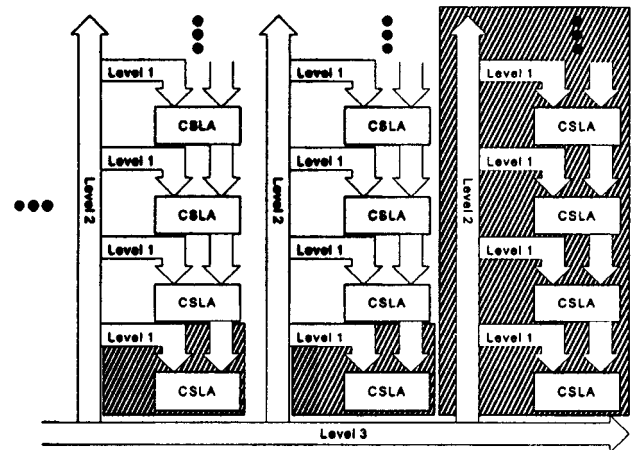


Fig. 6. Hierarchical redundancy.

unused in at least one configuration). If not, 2 pipe portions might have to be set aside for the array, or such a fault can be considered intolerable within the array.<sup>8</sup>

This problem can be resolved through a second tiling approach that involves looking at entire CSLA as the redundant block size. This can be achieved on the CSRC device by using the same steps as the 4-cell pipe portion level but applying them to larger areas: 4-cell pipe portions become CSLA, Level-1 routing becomes Level-2, etc. Configurations can be generated that leave 1 CSLA to be redundant for the active arrays in its 16-bit data pipe. The problems that existed at the 4-cell pipe portion level exist again at this next level, and they can be dealt with in the same manner. Level-2 interconnect can be reserved as unused in various contexts, just as Level-1 lines were, and the carry lines in and out of the arrays pose the same problem at this level of granularity and must be dealt with similarly. But, this approach solves the carry line problem at the logic cell level. If a fault occurs in a logic cell carry line, the entire array can be disabled as faulty. This approach also is more efficient if there are highly correlated faults that can render entire arrays faulty. The approach can also be taken up to the next level with pipes of CSLA being the redundant block size with the Level-3 routing and the carry lines in and out of the pipe posing the same problems. Again, redundancy at this next level solves the carry line problem from the previous level and recovers from *s*-correlated faults that can disable entire pipes.

Choosing the proper redundancy level can depend on many factors, including the application, the desired level of fault recovery (the number and types of faults to be tolerated), the amount of spare resources, and the acceptable amount of overhead (timing and area). Often, the best technique is a fractal-like hierarchical approach combining all the levels of redundancy levels. Within each logic array, there can be a redundant 4-cell pipe portion. Within each data pipe, there can be a redundant CSLA, and within each context, there can be a redundant data pipe (see Fig. 6). The CSRC device is inherently self-similar

<sup>8</sup>Considering such a fault intolerable under the given approach is a reasonable concession. The 4 carry lines occupy a relatively small area and, therefore, are less susceptible to faults than the other interconnect or logic which occupy a much larger area of the die.

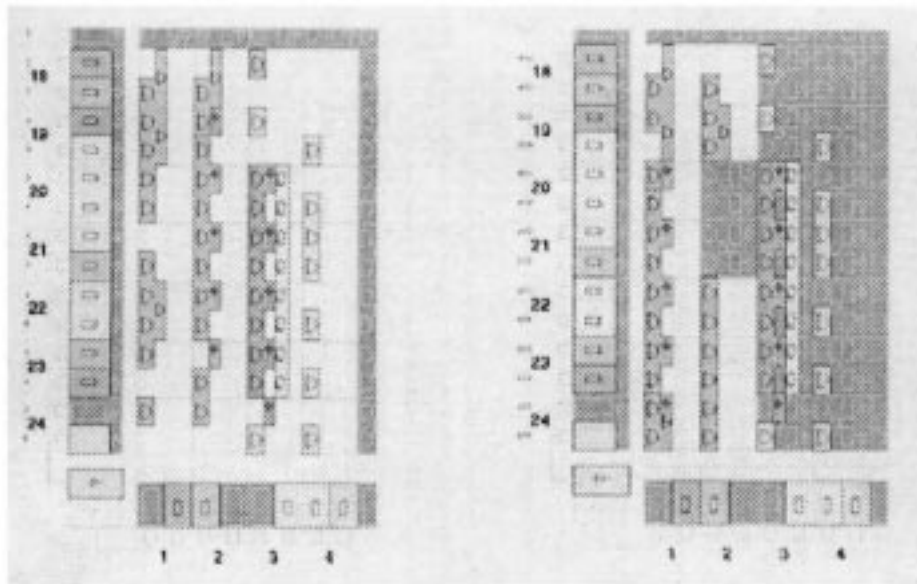


Fig. 7. PREP 5 before and after tilings with 1 AFTB identified.

and hierarchical in design and is therefore well suited for such an approach. This emphasizes the flexible nature of the general approach. That is, different levels of fault recovery and various acceptable amounts of overhead can be achieved for specific applications within the same general algorithm. Therefore, a wider array of fault models can be accommodated. Isolated single faults can be handled within individual arrays, because there is no need to render large sections of the chip useless for a small fault. Conversely, the higher levels can more efficiently tolerate a many  $s$ -correlated faults that could render entire arrays or even entire data pipes inoperable.

### B. Xilinx XC4000

Implementing the Algorithm on the Xilinx XC4000 family requires many alterations, but the general algorithm remains unchanged. The main architectural features that require the alterations are the nonfractal and nonhierarchical nature of the family and the wide use of segmented, overlapping interconnect. The former requires that the tiling algorithm be altered, and the latter requires the use of the inter-tile interconnect approach in Section I-B.

The logic in the 4000 family is not split into cells, arrays, and pipes as the CSRC device. Instead, there is simply a general array of CLB that, along with the local interconnect, can be tiled into smaller groups of CLB. Fig. 7 is an example of a small design (from the PREP benchmark set) implemented on the 4000 architecture at the floor-plan level. The figure on the left shows the original layout without any constraints imposed. From this layout, tile boundaries are established, and various AFTB are generated for each tile, constraining different resources as unused for each instance. The figure on the right shows the design after tiling (columns 1 and 2 and columns 3 and 4) and with 1 AFTB for 1 tile identified with 2 spare CLB (the other tile remains unaltered). AFTB generation continues until the proper level of fault-recoverability is achieved.

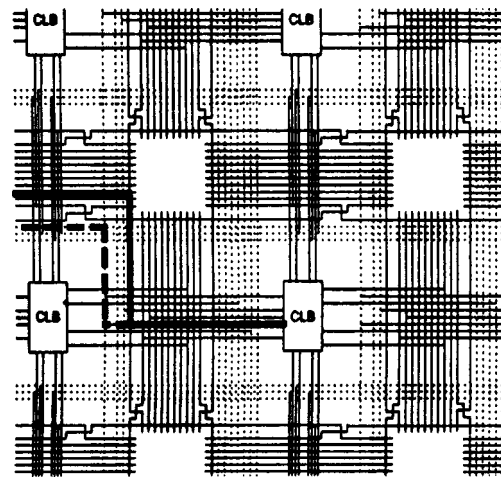


Fig. 8. Inter-tile interconnect fault-recovery on the Xilinx XC4000 architecture.

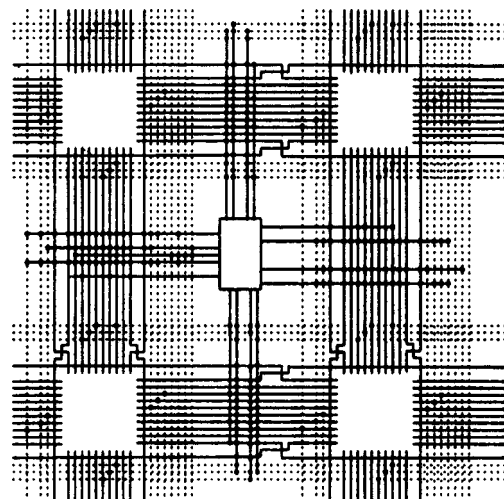


Fig. 9. Sample of interconnect connections available on XC-4000 family.

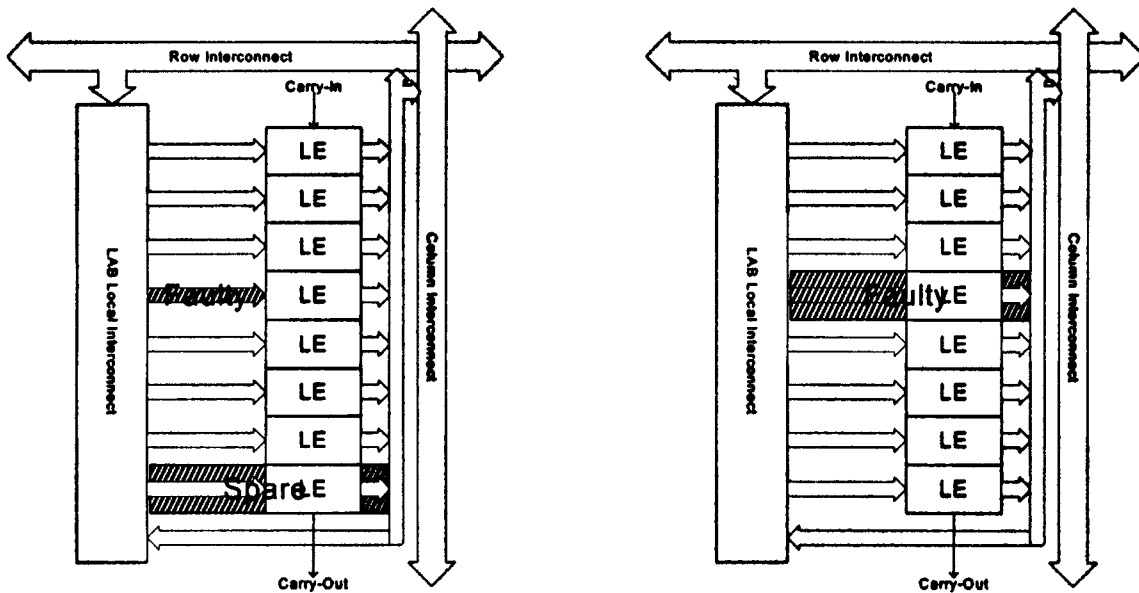


Fig. 10. Original and recovered configuration after an internal LAB fault.

The placement and shape of the tiles are determined by 3 key factors, listed in decreasing order of importance:

- 1) Amount of interconnect across the tile interface.
- 2) Tile logic density.
- 3) Tile size.

Tiling lines are drawn across areas with little inter-tile interconnect to ease the interface locking process and minimize the performance degradation. The logic density of each tile must allow some unused logic for redundancy and should be flexible and malleable to enable various configuration possibilities. Tile size is a factor, because large tiles can incur large overhead and low fault recovery levels as described in Section III-A. Tile size can vary from extremely fine-grained tiles of only 5–10 CLB to coarse-grained partitions containing over 50 CLB. Granularity is defined by the user, and the tiling algorithm partitions the design accordingly. If the first tiling attempt does not meet the user area or fault recovery specifications, the algorithm must repeat and find a different partition. Once tile lines are drawn, the *s*-independent tile implementation becomes quite similar to that for the CSRC device. Instances of each tile are generated at design-time that leave a portion of the tile (CLB and interconnect) unused. When a fault occurs, a tile instance can be activated that does not use the faulty resource.

The second change for implementation on the 4000 family involves the inter-tile interconnect. The 4000 architecture contains many lines that cross tile boundaries, and many are segmented and overlapped. Fig. 8 is an example of how the use of the inter-tile interconnect algorithm from Section I-B can be implemented on the 4000 architecture. The solid lines represent the segmented interconnect, and the dotted lines are the global lines. The solid bold line shows the signal before a fault is detected on one of the segmented lines, and the dotted line reflects a possible recovery from that fault using the global line.

The solution works theoretically on this architecture, but the current implementations do not allow the algorithm to work in practice. Although lines could be made available to backup

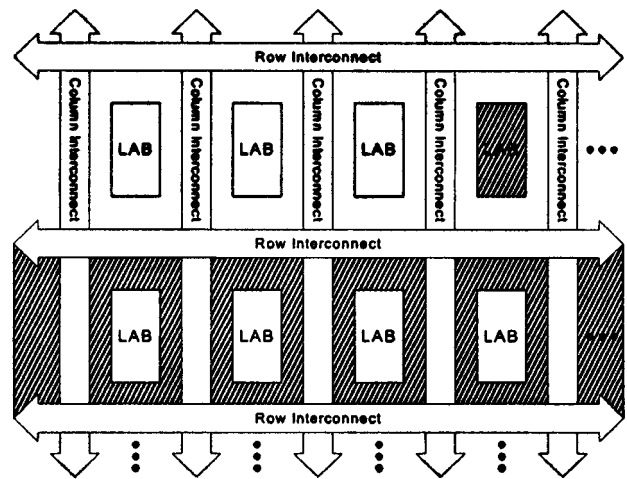


Fig. 11. Hierarchical redundancy.

TABLE I  
TIMING AND AREA OVERHEAD

Design	Slowest – Fastest	Final – Original
	Fastest	Original
9sym	0.21	.072
c499	0.25	.026
c880	0.17	.049
duke2	0.42	.077
rd84	0.45	.041
planet1	0.39	.056
styr	0.28	.039
s9234	0.41	.063
sand	0.26	.102

inter-tile lines, the connections do not currently exist for such an implementation, preventing the signals from making the proper routing alterations (see Fig. 9). The general architecture supports the approach, but not all connections were made possible in the implementation. There would not be an appreciable cost increase for adding more possible connections, making the

TABLE II  
RELIABILITY OF THE ORIGINAL AND TILED DESIGN AGAINST RESOURCE RELIABILITY

$p$	.900		.950		.990		.999		.9999	
	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled
9sym	0.78	10.99	7.54	36.53	47.5	91.8	91.7	99.5	96.8	100
c499	0.01	1.06	2.08	24.38	32.2	83.3	85.2	98.3	96.6	100
c880	0.00	0.39	1.17	20.61	28.7	85.2	84.8	98.1	97.1	100
duke2	0.01	0.46	1.82	20.74	29.6	84.7	85.2	98.4	96.5	100
rd84	4.68	25.09	18.01	48.56	58.9	91.3	92.6	99.2	97.8	100
planet1	0.01	3.51	7.48	21.19	28.8	89.6	85.7	98.0	96.1	100
styr	0.01	1.89	1.63	20.41	32.3	87.5	88.1	97.9	97.5	100
s9234	0.00	0.00	0.01	2.06	9.8	75.8	75.6	98.3	96.2	100
sand	0.02	0.99	1.19	1.63	31.4	85.2	83.5	98.6	96.9	100

eventual implementation of the algorithm on the architecture a possibility.

### C. Altera Flex 10k

The Altera Flex 10k family is more similar to the CSRC architecture than the Xilinx XC4000 family. The hierarchical structure returns, and the interconnect is more contained. Therefore, the implementation on this architecture is similar to that on the CSRC device.

The unit analogous to the CSLC is the LE, 8 of which comprise the logic for a LAB, which is analogous to the CSLA. The local interconnect within a LAB is also quite contained, allowing for a redundancy similar to that used in the CSLA, as it is structured like the Level-1 interconnect. One LE can be redundant for the others within the group of 8, and local interconnect can be set aside in each instance of the LAB generated at design-time by the CAD 0tools. Fig. 10 shows the internal structure of a LAB and how the block can recover from an LE (or associated interconnect) fault.

The hierarchical structure that efficiently implements the Flex 10k family as was done for the CSRC device also creates the same problems that existed for the CSRC device. LE have carry lines that cannot easily be made fault-tolerant within a single LAB. Fortunately, Flex 10k similarities to the CSRC device allow for similar solutions to be available. Therefore, the next level of redundant block size must be inspected, beginning the fractal-type approach for this architecture.

Groups of LAB form logic arrays, similar to the pipe level of the CSRC device, and the row or column interconnect is analogous to the Level-2 routing. One LAB can be redundant for the others within its pipe, and configurations can be generated for each LAB that might become faulty. Carry lines can flow in and out of pipes, potentially requiring that entire logic arrays also have a spare on the device, in the same way that pipes could have a spare on the CSRC device.

Using these levels of redundant blocks in combination on the Flex 10k architecture (Fig. 11 reveals the hierarchy) can yield the same benefits as those described for the CSRC architecture, including reduced area and timing overhead and increased levels of fault recovery.

## IV. EXPERIMENTAL RESULTS

The key factor to consider when evaluating this approach is the additional system reliability provided by its implementation.

TABLE III  
RELIABILITY OF ORIGINAL AND TILED DESIGN USING STAPPER'S CORRELATED FAILURE MODEL WITH RESOURCE RELIABILITY OF 90% AND A VARIABLE  $\mu$

$\mu$	1		5		20	
	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled
9sym	40.69	46.80	18.53	26.07	5.70	19.31
c499	37.64	44.33	12.94	23.53	1.79	9.43
c880	36.34	42.71	11.90	21.97	1.35	7.28
duke2	37.44	44.14	12.61	16.84	1.65	9.30
rd84	43.16	49.86	23.99	36.66	11.70	34.32
planet1	37.51	44.14	12.68	16.84	1.68	9.30
styr	38.29	44.27	14.04	25.74	2.36	10.01
s9234	34.52	41.86	8.52	18.46	0.41	3.46
sand	37.96	44.20	13.46	20.87	2.05	9.62

However, overhead in terms of area (physical resources) and timing must be considered to reveal its efficiency compared to traditional redundancy-based approaches to fault recovery. If a higher reliability is necessary, area and timing overhead might suffer. Conversely, if area and/or timing are strict system constraints, reliability must be sacrificed. The approach in this paper is flexible with respect to this reliability/overhead tradeoff.

For the Xilinx XC4000 family (the architecture least friendly to the implementation of the technique), the proposed approach and optimization algorithms were applied to 9 MCNC designs, with various logic and interconnect densities. Table I shows the timing and area metrics for the designs before and after the application of the fault recovery approach. Area overhead analysis compares the total design space of the tiled, fault-tolerant layout to the original, unconstrained layout.

Timing and area overhead are flexible, and Table I reveals only one possible implementation instance for each design. This instance was used for the reliability calculations in Tables II and III. A specific effort was made to minimize the area overhead for these implementations, requiring the need for small, fine-grained tiles. As a result, routing was constrained (causing higher timing overhead) more than it would be with larger tiles and a lesser emphasis on minimal area overhead. Again, the flexibility of tiling allows the user to prioritize overhead and reliability for specific applications.

Table II shows reliability improvements for the MCNC benchmarks under a uniform random fault model.<sup>9</sup> The top row indicates the assumed probability ( $p$ ) that a physical element (logic or interconnect) is fault free for the lifetime of

<sup>9</sup>Not all fault combinations were considered, because the XC4000 does not allow for all of the connections required by inter-tile interconnect recovery.

the system. The left column gives the name of the design. The 2 columns for each  $p$  show the probability that the original and fault recoverable designs of a particular benchmark are functioning properly. Table III shows the reliability for the same set of designs (original and tiled) for Stapper's correlated fault model [20], with 4 different cluster variability factors (degree of clustering),  $\mu$ , assuming a 90% fault free probability for physical resources.<sup>10</sup>

The same evaluations can be performed on the Sanders CSRC and Altera Flex 10k architectures. The analysis in Sections III-A and III-C shows that the hierarchical nature and minimal segmented overlapped interconnect of the CSRC architecture and the Altera's Flex 10k family create even greater opportunities for efficient implementation. Therefore, these results reveal a worst-case analysis in terms of applying the approach to the diverse architectures examined in this paper.

#### REFERENCES

- [1] A. Dasgupta and R. Karri, "High-reliability, low-energy microarchitecture synthesis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1273–1280, Dec. 1998.
- [2] K. Kim, R. Karri, and M. Potkonjak, "Configurable spare processors: A new approach to system level fault-tolerance," in *Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1996, pp. 295–303.
- [3] R. Karri, K. Hogstedt, and A. Orailoglu, "Computer-aided design of fault-tolerant VLSI systems," *IEEE Design and Test of Computers*, vol. 13, no. 3, pp. 88–96, 1996.
- [4] A. Orailoglu and R. Karri, "Automatic synthesis of self-recovering VLSI systems," *IEEE Trans. Computers*, vol. 45, no. 2, pp. 131–142, Feb. 1996.
- [5] A. L. Burrell and P. K. Lala, "On-line testable logic design for FPGA implementation," in *Int. Test Conf.*, 1997, pp. 471–478.
- [6] W. Feng, W. K. Huang, and F. Lombardi, "Structural testing of programmable interconnects," *J. Microelectronic Systems Integration*, vol. 5, no. 3, pp. 129–144, Sept. 1997.
- [7] C. Metra *et al.*, "Novel technique for testing FPGAs," *Design, Automation, and Test in Europe*, pp. 89–94, 1998.
- [8] M. Nicolaidis, "On-line testing for VLSI: State of the art and trends," *Integration, the VLSI J.*, vol. 26, no. 12, pp. 197–209, Dec. 1998.
- [9] M. Renovell *et al.*, "RAM-based FPGAs: A test approach for the configurable logic," *Design, Automation and Test in Europe*, pp. 82–88, 1998.
- [10] M. Renovell *et al.*, "Testing the interconnect of RAM-based FPGAs," *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 45–50, Jan.–Mar. 1998.
- [11] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *IEEE Trans. VLSI Systems*, vol. 6, no. 2, pp. 212–221, 1998.
- [12] G. A. Allan and A. J. Walton, "Automated redundant via placement for increased yield and reliability," in *Proc. SPIE—The Int. Society for Optical Engineering*, vol. 3216, Microelectronic Manufacturing Yield, Reliability, and Failure Analysis III, 1997, pp. 114–125.
- [13] F. Distanto, M. G. Sami, and R. Stefanelli, "Harvesting through array partitioning: A solution to achieve defect tolerance," in *Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1997, pp. 261–269.
- [14] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: Techniques and yield analysis," in *Proc. IEEE*, vol. 86, Sept. 1998, pp. 1819–1838.
- [15] S. Dutt, V. Shanmugavel, and S. Trimberger, "Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays," in *Int. Conf. Computer-Aided Design*, 1999, pp. 173–176.
- [16] M. Abramovici *et al.*, "Using roving STAR's for on-line testing and diagnosis of FPGA's in fault-tolerant applications," in *Int. Test Conf.*, 1999, pp. 973–982.
- [17] S. M. Scalera and J. R. Vázquez, "The design and implementation of a context switching FPGA," in *Symp. FPGA-Based Custom Computing Machines*, 1998, pp. 78–85.
- [18] Xilinx, *The Programmable Logic Data Book*, 1996.
- [19] Altera, *Data Book*, 1996.
- [20] C. H. Stapper, "A new statistical approach for fault-tolerant VLSI systems," in *Int. Symp. Fault-Tolerant Computing*, 1992, pp. 356–365.

**John Lach** received the B.S. (1996) from Stanford University, and the M.S. (1998) and Ph.D. (2000) from UCLA. He is an Assistant Professor in the Electrical Engineering Department at the University of Virginia. His research interests include algorithms for VLSI computer-aided design (CAD) tools, application specific and general purpose processor design, back-end compilers, and intellectual property protection.

**William H. Mangione-Smith** attended the University of Michigan, Ann Arbor, where he received a B.S.E. (1987) in electrical engineering, and M.S.E. (1992) and Ph.D. (1992) in computer science and engineering. 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 Associate Professor in the Electrical Engineering Department at UCLA. He is a Co-PI on the Mojave configurable computing program and served as the Program Chair'n for MICRO 26. 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.

**Miodrag Potkonjak** received the Ph.D. (1991) in electrical engineering and computer science from University of California, Berkeley. In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, and UCLA in 1995. He is a Professor in the Computer Science Department at UCLA, and has received the NSF CAREER award, the OKAWA foundation award, the UCLA TRW SEAS Excellence in Teaching Award, and several best-paper awards. He has published about 200 papers in leading CAD and VLSI design, real-time systems, multimedia, signal processing, computational security, and communications journals and conferences. He holds 5 patents. He has been serving on several Program Committees, including International Conference on Image Processing, Design Automation Conference, and International Conference on Computer-Aided Design. His recent watermarking-based intellectual property protection (IPP) research formed a basis for the Virtual Socket Initiative Alliance (IPP) standard. His research interests include embedded systems, sensor networks, computational security, and intellectual property protection.

<sup>10</sup>Detailed reliability equations for  $s$ -independent and clustered fault models are in [11]. Reliability numbers here include analysis of interconnect fault recoverability.