Runtime Logic and Interconnect Fault Recovery on Diverse FPGA Architectures

J. Lach[1], W. H. Mangione-Smith[1], and M. Potkonjak[2]
[1]UCLA EE Department, 56-125B Engineering IV, Los Angeles, CA 90095 {jlach, billms}@icsl.ucla.edu
[2]UCLA CS Department, 3532G Boelter Hall, Los Angeles, CA 90095 miodrag@cs.ucla.edu

Abstract

Field programmable gate arrays (FPGAs) have recently graduated from being applied almost solely to prototyping, logic emulation systems, and extremely low volume applications, and they are now used in a number of high-volume consumer devices. The inherent redundancy and in-the-field reconfiguration capabilities of FPGAs provide alternatives to integrated circuit redundancy-based fault recovery techniques. An algorithm for efficient runtime recovery from permanent logic faults in the Xilinx 4000 has been expanded to include interconnect fault recovery and has been applied to a diverse set of FPGA architectures. The post-fault-detection system downtime is minimized, and the end user need not have access to computer-aided design (CAD) tools, making the algorithm completely transparent to system users. Although some architectural features allow for a more efficient implementation, high levels of fault recovery with low timing and resource overhead can be achieved on these diverse architectures.

I. Introduction

Taking advantage of the flexible and inherently redundant nature of FPGAs, a low overhead fault recovery algorithm has been developed capable of recovering from faults at runtime with minimal system downtime and no end user CAD tool requirements. Assuming a detection[1], localization, and diagnosis[2] 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. Previous algorithms achieved such runtime fault recovery exclusively for logic faults [7], but the algorithm has been expanded to include interconnect faults and has been applied to a variety of FPGA architectures.

[1] Many FPGA testing techniques are currently available for both logic and interconnect [1-6].

[2] Any fault that occurs can be considered permanent and therefore further use of the faulty resource can be prevented. Therefore, diagnosis is not entirely necessary, but it may be helpful in identifying non-permanent faults that can be corrected, allowing further use of the faulty resource.

A. General Algorithm

We propose partitioning the physical design into a set of tiles. Each tile is composed of a set of physical resources (i.e. logic blocks and interconnect), an interface specification which denotes the connectivity to neighboring tiles, and a netlist. Logic and local interconnect reliability is 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 independent and reducing the storage overhead[3]. Any paths that cross tile boundaries can be made reliable by reserving other inter-tile interconnect to be used as spares.

This approach has three main benefits compared to redundancy-based fault recovery techniques: very low overhead, the option for runtime management, and flexibility. The overhead required to implement this fine-grained approach, which can be measured in both physical resources on the FPGA (logic blocks, I/O blocks, and interconnect) and circuit performance, is extremely low compared to redundancy. This is due primarily to the inherent redundancy in FPGAs, as opposed to the introduction of redundant elements into fixed designs that is required for reliability and yield enhancement [8-10]. Runtime management can be a very valuable feature 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 flexibility that this approach provides allows for application-specific solutions. The degree of fault recovery can be adjusted to meet timing constraints, resource limitations, or estimated logic block and interconnect reliability.

B. Example

Consider the Boolean function $Y=(A \wedge B) \wedge (C \vee D)$, which might be implemented in a tile containing four logic blocks as shown in the Figure 1. This partitioning contains one spare logic block, which is available if a fault should be detected in one 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, as the interface between the tile and the surrounding areas of the design is fixed and the

[3] Tile 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.

individual configurations implement the same function. The timing of the circuit may vary, however, due to the changes in routing.

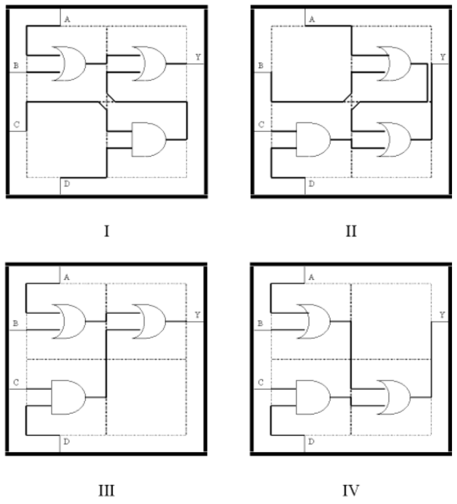

Figure 1: Logic fault recovery

Recovering from local interconnect faults can be handled in much the same manner. Interconnect can be set aside as unused in each tile configuration[4], and when a fault disables an interconnect line an instance not utilizing that line can be activated.

Recovering from faults to global and overlapped segmented interconnect requires a different approach, as much of that interconnect crosses tile boundaries, thus eliminating the 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. Figure 2 shows how a global line (dotted) can act as a backup for several segmented interconnect lines (solid) in the Xilinx XC4000 family. Upon a segmented line sustaining a fault, 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. Again, the only difference may be in the timing of the circuit.

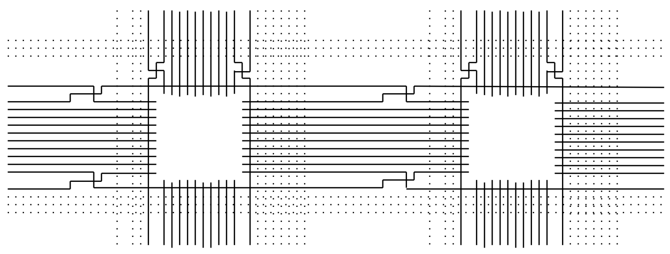

Figure 2: Inter-tile interconnect fault recovery

[4] 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.

Tiling provides many advantages in the implementation of fault recovery FPGA systems. First, the amount of place-and-route effort and memory needed to generate and store the set of configurations is smaller than the amount required to generate and store a set of complete configurations. For example, consider a design that must be able to recover from any single logic block fault and which maps into a 6x6 logic block array. It may be possible to divide the design into four 3x3 tiles. Assuming that one configuration of the complete 6x6 design requires *X* amount of effort and *X* bytes of memory, the non-tiled approach would require *36*X* of effort and memory for fault recovery capabilities: one complete design configuration for each logic block that is at risk. With our method, each tile would require 9 configurations. However, since each tile (X/4 effort and storage bytes) is independent, the entire effort and storage is only *9*X*, a 75% reduction from the non-tiled approach.

Tiling also increases reliability. For this example, the non-tiled approach could recover from only one faulty logic block in the entire device. The tiled approach, however, is capable of recovering from any single fault in a tile but up to 4 faulty logic blocks in the entire device.

The cost of increased fault recovery and reduced configuration memory is the possible introduction of more spare resources. For this example, the non-tiled approach reserves 2.7% of the logic blocks to protect against a single fault, while the tiled approach reserves 11%. However, tiling opens up the opportunity to explore a rich design space. By choosing the tile size and amount of spare resources that are appropriate for system requirements, tiling provides a powerful and flexible tool to the designer.

C. *Motivation and Fault-Models*

There are two major sources of logic faults in FPGA systems: cosmic radiation and manufacturing/operating imperfections, both of which can affect logic and interconnect [11].

Since the size of radiation particles is usually small when compared to the size of a modern FPGA logic block, we selected a cosmic radiation fault model that follows uniform distribution of independent (uncorrelated) failures. Extensive terrestrial efforts to accurately model the rate of such soft faults indicate high variance (several orders of magnitude) depending on factors such as 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 [12]. Experiments indicate that in FPGA-like devices at an altitude of 20 km, error rates significantly higher than once per 1000 hours are common [13]. Also, as circuit devices become smaller, they become more sensitive to soft faults [12]. Radiation error rates in specific current FPGA technologies have also been calculated [14]. If one considers the multiyear life of computing devices and other sources of potential errors (e.g. power surges), the need for fault recovery in devices which implement critical functions becomes apparent. However, different applications may require different degrees of fault recovery, requiring a flexible fault-tolerance algorithm and implementation [15].

The 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. 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). For this type of model, we follow the gamma-distribution Stapper fault model [16]. The model is applicable on any integrated circuit with regular repetitive structure, including memories and FPGA devices.

II. FPGA ARCHITECTURES

The implementation of the algorithm varies depending on the target FPGA architecture. The following sections describe the implementation on three architectures: Sanders' context switching reconfigurable computing (CSRC) technology [17], Xilinx's XC4000 family [18], and Altera's Flex 10k series [19].

A. Sanders CSRC

The CSRC device is based on new architectural techniques that exploit dynamic reconfiguration via context switching. Each context is identical, and a cross-context sharing mechanism enables data sharing between contexts. The device can switch contexts in a single clock, and non-active contexts can be loaded and configured in the background of active context execution. Each context is composed of logic and interconnect arranged in a hierarchical structure. At the lowest level of logic is the context switching logic cell (CSLC). Sixteen CSLCs comprise one context switching logic array (CSLA), as shown in Figure 3. Each logic cell has a carry-in and carry-out. The sixteen CSLCs are sectioned into four groups that are connected and driven by Level 1 routing.

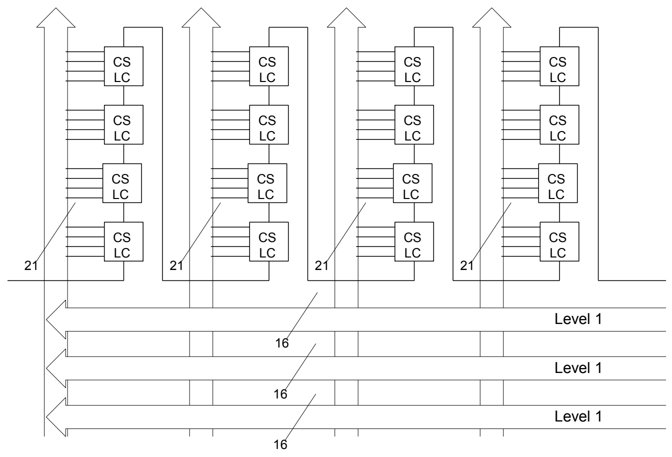

Figure 3: CSLA and Level 1 routing

The next level of logic and interconnect is the 16-bit data pipe which is composed of CSLAs and connected by Level 2 routing as shown in Figure 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.

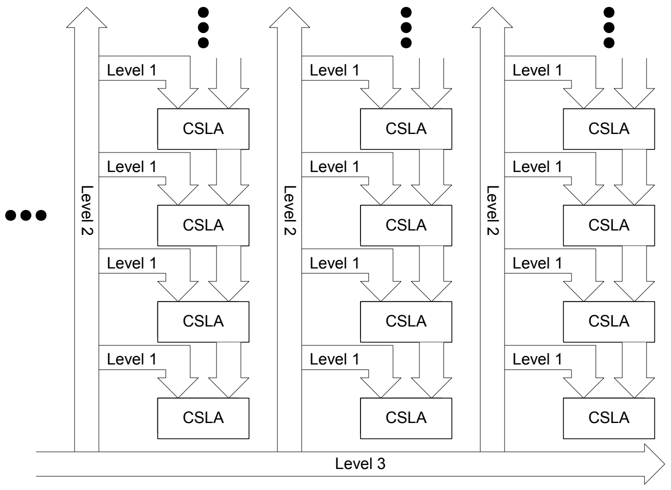

Figure 4: Data pipes and Level 3 routing

Developing a fault recovery approach for the CSRC device can be done in two ways: 1) looking at each context as an independent entity, and 2) allowing functionality belonging to one context to be transferred to another in the face of a fault. Analysis reveals that the latter 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 is capable of performing 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 in the case of 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[5]) and only be able to recover from X faults throughout the lifetime of the system. Fault recovery in FPGAs 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 would be analogous to simply adding redundant FPGAs in a non-CS system.

Looking at each context as an independent entity, a much finer grained approach, helps to alleviate these problems. For example, looking at smaller blocks of redundancy (CSLAs or even CSLCs) creates the opportunity for lower resource overhead and a smaller negative performance impact. CAD tools rarely map logic to maximum density, leaving many CSLAs and CSLCs unused even in a non-fault-tolerant configuration. 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

[5] Sanders' CSRC device has four contexts. With three active contexts and one spare, resource overhead would be 25% while being capable of tolerating only one fault.

distributed for easier fault-recovery and a smaller performance impact, but the overhead is still significantly 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 (see Section I.C). 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, as one faulty wire or LUT should not render an entire context faulty.

Breaking the CSRC device contexts into independent fault recovery blocks (tiling) can be done in a number of 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 Figure 3). If a logic cell in the middle of a 4-cell pipe portion should fail, it may 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, i.e. the 4-cell pipe portion. Each portion has the same connectivity, making it possible for one pipe portion to be redundant for any other in the array. Thus, each array has one 4-cell pipe portion that is redundant for the other three, and three other configurations are generated that would be instantiated upon a failure to one 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 array configuration not utilizing the pipe portion containing the faulty hardware recovers from the fault. Figure 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 use. Therefore, configurations must be generated at design-time that do not make use of each Level 1 line in at least one configuration, much in the same way that each pipe-portion is unused in at least one configuration.

Recovering from faults to the carry line is more difficult, as such a fault potentially renders two 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 one pipe portion and the carry-in of another are faulty, an inter-pipe portion problem arises. The CAD tool may be able to implement the array's functionality without using the carry line (i.e. each carry line unused in at least one configuration). If not, two pipe portions may have to be set aside for the array, or such a fault may be considered intolerable within the array[6].

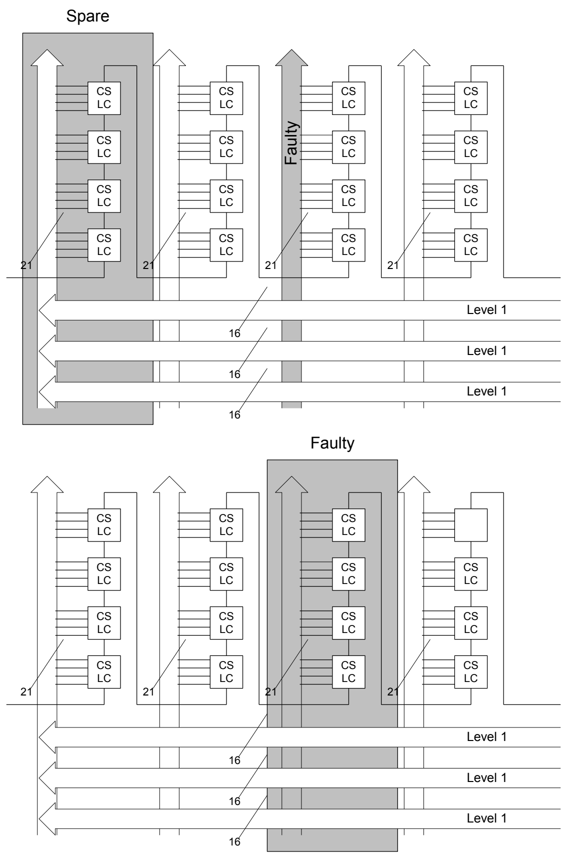

Figure 5: Original and recovered configuration after an internal CSLA fault

This problem can be resolved through a second tiling approach that involves looking at entire CSLAs as the redundant block size. This can be achieved on the CSRC device by using the exact same steps as the previous approach but simply applying them to larger areas: 4-cell pipe portions become CSLAs, Level 1 routing becomes Level 2, etc. Configurations can be generated that leave one CSLA to be redundant for the active arrays in its 16-bit data pipe (see Figure 4). The problems that existed for the first approach 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 as before 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 (see Section I.C) which may render entire arrays faulty.

[6] Considering such a fault intolerable under the given approach is a reasonable concession. The four 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.

The approach can also be taken up to the next level with pipes of CSLAs being the redundant block size with the Level 3 routing and the carry lines in and out of the pipe posing the same problems. But again, redundancy at this next level solves the carry line problem from the previous level and recovers from correlated faults that may disable entire pipes.

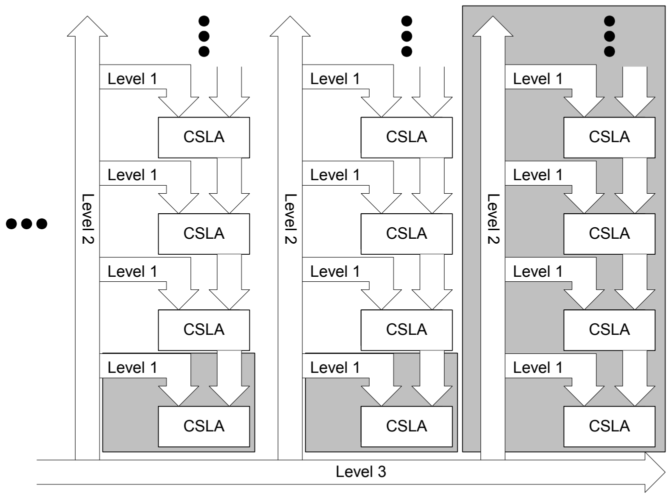

Figure 6: Hierarchical redundancy (shaded areas are spares)

Choosing the proper level of redundancy may 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 times, the best technique will be a fractal-like approach combining all of the above 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 Figure 6). It can even be taken to the extreme addressed above in which each device contains a redundant context[7]. Although such an extreme may not be efficient to implement or even be practical within the given fault model, a lower level fractal-like approach may be most desirable for many reasons. First, the CSRC device is inherently self-similar and hierarchical in design and is therefore well suited for such an approach. Second, it 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. Third, each consecutive level solves problems that arise at the previous (e.g. faults to the carry lines). Finally, a wider array of fault models can be accommodated. Isolated single faults can be handled within individual arrays, as 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 large number of correlated faults that may render entire arrays or even entire data pipes inoperable.

[7] A problem does arise when considering a level between redundant data pipes and redundant contexts, as such a level would require tolerating faults to the cross-context data sharing mechanism. The current proposed approach does not tolerate faults to this mechanism.

The CSRC architecture is well suited for the proposed fault recovery approach. Tiling lines can be drawn at many levels, enhancing the approach's flexibility and thereby increasing the level of fault recovery (both number and types of faults) while decreasing area and timing overhead.

B. Xilinx XC4000

Implementing the same 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 non-fractal and non-hierarchical 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 described in Section I.B.

The logic in the 4000 family is not broken up into cells, arrays, and pipes as the CSRC device. Instead, there is simply a general array of configurable logic blocks (CLBs) which, along with the local interconnect, can be tiled into smaller groups of CLBs. Figure 7 shows an example of a small design (from the PREP benchmark set) implemented on the 4000 architecture. The first shows the original layout, and the second shows the design after tiling and with one configuration for one tile identified with two spare CLBs.

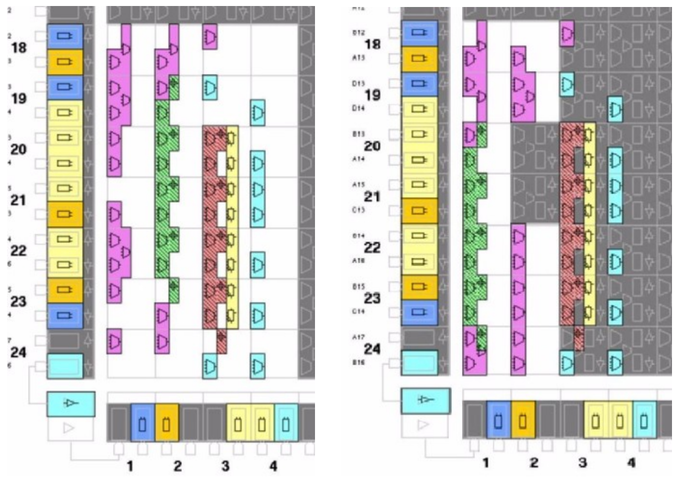

Figure 7: PREP 5 before and after tiling with one tile configuration identified

The placement and shape of the tiles are determined by three key factors listed in decreasing order of importance: amount of interconnect across the tile interface, tile logic density, and tile size. Tile lines are drawn across areas with little inter-tile interconnect to ease interface locking 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 also a factor, as large tiles may incur large overhead and low fault recovery levels as described in Section II.A. If a tiling attempt does not meet the user area or fault recovery specifications, the algorithm must repeat and find a different tile partition. Once tile lines are drawn, the 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 (CLBs and interconnect) unused. When a fault occurs, a tile instance can be activated that does not utilize 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. Figure 8 shows an example of how the use of the inter-tile interconnect algorithm from Section I.B could be implemented on the 4000 architecture. The solid lines represent the segmented interconnect, and the dotted are the global lines. The bold line shows the signal before and after a fault is detected on one of the segmented lines and the signal switches to a backup global line.

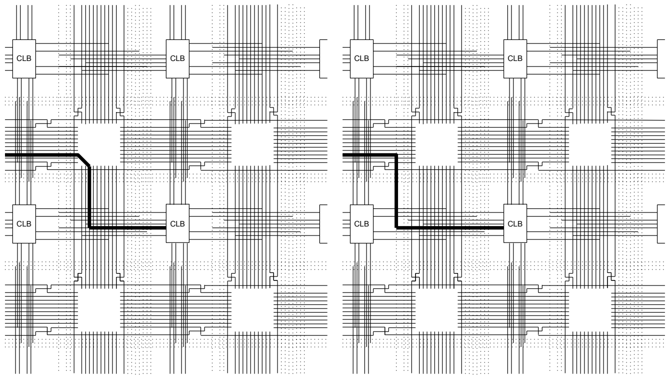

Figure 8: Inter-tile interconnect fault recovery on the Xilinx XC4000 architecture

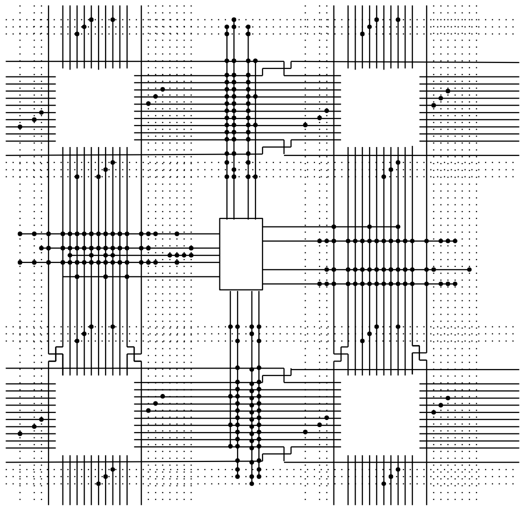

Figure 9: Sample of interconnect connections available on XC4000 family

The solution works theoretically on this architecture, but the current implementations would not allow the algorithm to work in practice. Although lines could be made available to backup inter-tile lines, the connections do not currently exist for such an implementation, preventing the signals from making the proper routing alterations (see Figure 9). The general architecture clearly supports the approach, but such connections where thought to be unnecessary in the specific 4000 family implementation. The only cost of adding such connections would be additional configuration bits, making the eventual implementation of the algorithm on the architecture a possibility. The one feature of the Xilinx family that eases algorithm implementation involves the absence of inherent carry lines. This benefit is outweighed, however, by the inter-tile interconnect dilemma.

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 logic element (LE), eight of which comprise the logic for a logic array block (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 eight, and local interconnect can be set aside in each instance of the LAB generated at design-time by the CAD tools. Figure 10 shows the internal structure of a LAB and how the block can recover from an LE (or associated interconnect) fault.

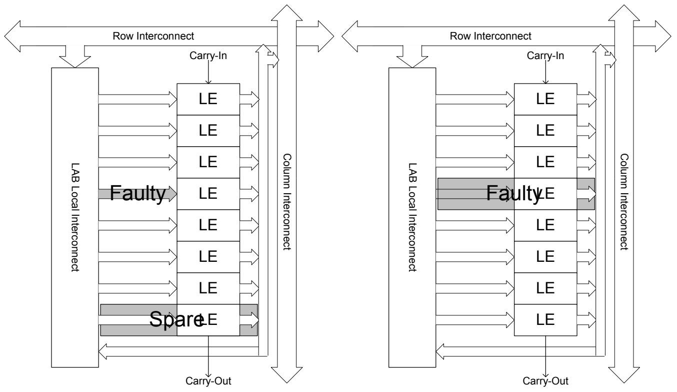

Figure 10: Original and recovered configuration after an internal LAB fault

The hierarchical structure that makes for an efficient implementation on the Flex 10k also creates the same problems that existed for the CSRC device. LEs 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 LABs form logic arrays, similar to the pipe level of the CSRC device, and the row or column[8] 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 may absorb a fault. Again, carry lines may 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.

[8] Row or column interconnect can be used for the analogy, depending on the direction the pipe is laid. Row interconnect is appropriate if the pipes flow horizontally. In such a case, the column interconnect then becomes analogous to Level 3 routing on the CSRC.

Using these levels of redundant blocks in combination on the Flex 10k architecture can produce the same benefits as those described for the CSRC architecture, including reduced area and timing overhead and increase levels of fault recovery. Figure 11 reveals the potential hierarchical structure of redundant blocks for the Flex 10k architecture.

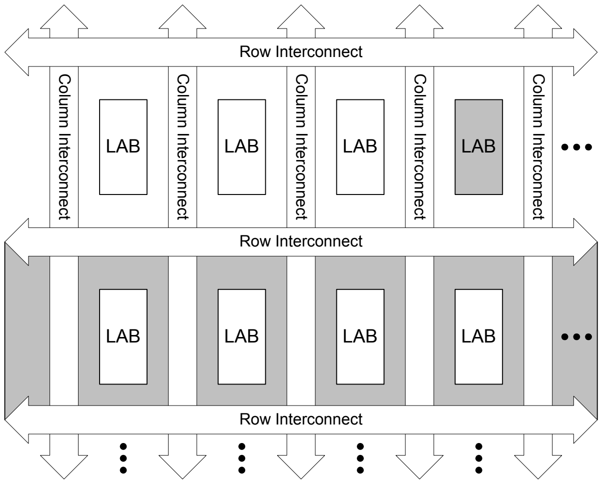

Figure 11: Hierarchical redundancy (shaded areas are spares)

III. Experimental Results

The key factor to consider when evaluating this approach is the additional reliability provided the system by its implementation. 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 may suffer. Conversely, if area and/or timing is a strict system constraint, reliability must be sacrificed. The approach presented in the 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 nine MCNC designs, with various logic and interconnect densities. Table 1 shows the timing and area metrics for the designs before and after the application of the fault recovery approach. This overhead is flexible, and the values in Table 1 only reveal one possible implementation instance. This instance was used for the reliability calculations displayed in Tables 2 and 3.

Table 1: Timing and area overhead

Design	$\frac{\text{Slowest - Fastest}}{\text{Fastest}}$	$\frac{\text{Final – Original}}{\text{Original}}$
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

Table 2 shows reliability improvements for the MCNC benchmarks under the uniform random fault model. For various assumed probabilities (p) that physical elements (logic or interconnect) are fault free, the probability of the original and fault recoverable versions of the benchmarks being functional is shown. Table 3 shows the reliability figures for the same set of designs (original and tiled) with four different cluster variability factors, μ, assuming the probability that a physical resource is fault free is 90%.

Table 2: Reliability of the original and tiled designs against resource reliability

p	.900		.950		.990		.999		.9999	
Design	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

Table 3: Reliability of original and tiled designs using Stapper's correlated failure model with resource reliability of 90% and a variable μ

	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

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

IV. Conclusions

Efficiently implementing a runtime fault recovery algorithm on a variety of architectures shows the flexible nature of the algorithm and reveals architectural features that enhance or hinder the approach. Runtime implementation of the algorithm on each architecture is straightforward and minimizes system downtime, and the approach minimizes the amount of memory and CAD tool effort required. Experimental results reveal that the area and time overhead remain low and that the fault recoverability is high even for the Xilinx architecture, the most difficult on which to implement the approach.

Acknowledgements

This work was supported by the Air Force Research Laboratory of the United States of America, under contract F30602-96-C-0350 and subcontract QS5200 from Sanders, a Lockheed Martin company.

References

[1] A. L. Burress, P. K. Lala, "On-line Testable Logic Design for FPGA Implementation", *International Test Conference*, pp. 471-478, 1997.

[2] W. Feng, W. K. Huang, F. Lombardi, "Structural Testing of Programmable Interconnects", *Journal of Microelectronic Systems Integration*, vol. 5, no. 3, pp. 129-144, Sept. 1997.

[3] C. Metra *et al.*, "Novel Technique for Testing FPGAs", *Design, Automation and Test in Europe*, pp. 89-94, 1998.

[4] M. Nicolaidis, "On-Line Testing for VLSI: State of the Art and Trends", *Integration, The VLSI Journal*, vol. 26, no. 1-2, pp. 197-209, Dec. 1998.

[5] M. Renovell *et al.*, "RAM-Based FPGAs: A Test Approach for the Configurable Logic*", Design, Automation and Test in Europe*, pp. 82-88, 1998.

[6] M. Renovell *et al.*, "Testing the Interconnect of RAM-Based FPGAs", *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 45-50, Jan.-March 1998.

[7] J. Lach, W.H. Mangione-Smith, M. Potkonjak, "Low Overhead Fault-Tolerant FPGA Systems", *IEEE Transactions on VLSI Systems*, vol. 6, no. 2, pp. 212-221, 1998.

[8] G. A. Allan, A. J. Walton, "Automated Redundant Via Placement for Increased Yield and Reliability", *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 3216 (Microelectronic Manufacturing Yield, Reliability, and Failure Analysis III), pp. 114-125, 1997.

[9] F. Distante, M. G. Sami, R. Stefanelli, "Harvesting Through Array Partitioning: A Solution to Achieve Defect Tolerance", *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 261-269, 1997.

[10] I. Koren, Z. Koren, "Defect Tolerance in VLSI Circuits: Techniques and Yield Analysis", *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1819-1838, Sept. 1998.

[11] M. Renovell *et al.*, "SRAM-Based FPGAs: A Fault Model for the Configurable Logic Modules", *Field-Programmable Logic and Applications*, pp. 139-148, 1998.

[12] J.F. Ziegler *et al.*, "IBM Experiments in Soft Fails in Computer Electronics (1978-1994*)*", *IBM Journal of Research and Development*, vol. 40, no.1, pp. 3-18, 1996.

[13] T.J. O'Gorman *et al.*, "Field Testing for Cosmic Soft-Error Rate", *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 51-72, 1996.

[14] R. Katz *et al.*, "Radiation Effects on Current Field Programmable Technologies", *IEEE NSREC/Transactions on Nuclear Science*, vol. 44, no. 6, pt. 1, pp. 1945-1956, 1997.

[15] L. Dadda, V. Piuri, "Bit-Modular Defect/Fault-Tolerant Convolvers", Proceedings the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 90-98, 1995.

[16] C. H. Stapper, "A New Statistical Approach for Fault-Tolerant VLSI Systems", *The Twenty Second International Symposium on Fault-Tolerant Computing*, pp. 356-365, 1992.

[17] S.M. Scalera, J.R. Vázquez, "The Design and Implementation of a Context Switching FPGA", *6th IEEE Symposium on FPGA-Based Custom Computing Machines*, 1998.

[18] Xilinx, The Programmable Logic Data Book, San Jose, CA, 1996.

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