Efficiently Supporting Fault-Tolerance in FPGAs

John Lach
UCLA EE Department
56-125B Engineering IV
Los Angeles, CA 90095
(310) 794-1630
jlach@icsl.ucla.edu

William H. Mangione-Smith
UCLA EE Department
56-125B Engineering IV
Los Angeles, CA 90095
(310) 206-4195
billms@icsl.ucla.edu

Miodrag Potkonjak
UCLA CS Department
4532K Boelter Hall
Los Angeles, CA 90095
(310) 825-0790
miodrag@cs.ucla.edu

1. ABSTRACT

While system reliability is conventionally achieved through component replication, we have developed a fault-tolerance approach for FPGA-based systems that comes at a reduced cost in terms of design time, volume, and weight. We partition the physical design into a set of tiles. In response to a component failure, we capitalize on the unique reconfiguration capabilities of FPGAs and replace the affected tile with a functionally equivalent tile that does not rely on the faulty component. Unlike fixed structure fault-tolerance techniques for ASICs and microprocessors, this approach allows a single physical component to provide redundant backup for several types of components. Experimental results conducted on a subset of the MCNC benchmarks demonstrate a high level of reliability with low timing and hardware overhead.

1.1 Keywords

FPGA, fault-tolerance

2. INTRODUCTION

2.1 Motivation

While once FPGAs were mostly applied to prototyping, logic emulation systems and extremely low volume applications, they now are used in a number of high volume consumer devices. FPGAs are also now being used in more exotic applications. For example, the Mars Pathfinder mission launched in 1996 by NASA relies on Actel FPGAs for some system services. Unlike early applications, these high volume and mission critical systems tend to have stringent reliability requirements [19]. Thus, there is a drive from the user community to improve reliability through some level of fault-tolerance.

Unfortunately, current technology trends tend to make FPGAs less reliable. FPGA vendors have been moving down the same path of smaller device size as the rest of the semiconductor industry. Electronic current density in metal traces will increase as device feature size shrinks from 0.5 um to 0.35 um and smaller, which results in a greater threat of electromigration. As transistors shrink, the amount of charge required to turn them on reduces, which also makes the components more susceptible to gamma particle radiation. At the same time, FPGA vendors are moving to larger and larger dies in order to deliver more logic gates to their customers. The larger dies introduce more opportunities for failure and bigger targets for gamma particles.

Engineers traditionally respond to these threats through redundancy, such as replicating components (e.g. microprocessors and ASICs) or replicating logic internal to a component (e.g. Built-In Self-Repair (BISR)). However, replication is a particularly unattractive approach for FPGA systems given the common customer complaint that devices cost too much and do not provide enough equivalent logic gates. A better approach is to leverage the flexible nature of FPGA devices to provide replication at a much finer level. Conceptually, if a single logic block fails, it is often possible to find an alternate circuit mapping that avoids the fault. Most vendor place and route tools provide an option for reserving resources, and in the face of a fault, the tool could be invoked to search for a new placement which only uses functional components. The resulting system could provide reliability with very low overhead (i.e. by reserving only a few percent of the resources as spares for fault recovery). Unfortunately, this approach results in significant system downtime. Thus, the technique will not be sufficient for mission-critical applications with hard real-

Presented at FPGA'98

Monterey CA, February 22-24, 1998

time constraints. This approach also requires that the end user have the vendor place and route tools, which is usually not possible. It seems unlikely that the end consumer will wish to even know about an embedded FPGA, let alone worry about generating a new configuration for one. Finally, because each fault is distinct, each component would possibly require a unique circuit placement. These three factors combine to make the approach impractical.

We instead propose a technique for increasing FPGA system reliability with very low overhead. The target architecture for demonstration is a Xilinx 4000EX part, which is composed of an array of configurable logic blocks (CLBs). Nonetheless, we believe that this technique is applicable to a wide range of FPGA architectures. The place-and-route CAD tool maps a circuit net-list onto the array of CLBs and interconnect components. We propose partitioning the physical design into a set of tiles. Each tile is composed of a set of physical resources (i.e. CLBs and interconnect), an interface specification which denotes the connectivity to neighboring tiles, and a net-list. Reliability is achieved by providing multiple configurations of each tile. Furthermore, by using locked tile interfaces, the effects of swapping a tile configuration do not propagate to other tiles, thus reducing the storage overhead.

2.2 Motivational Example

Consider the Boolean function $Y=(A \wedge B) \wedge (C \vee D)$, which might be implemented in a tile containing four CLBs as shown in Figure 1. This configuration contains one spare CLB, which is available if a fault should be detected in one of the occupied CLBs. Upon detecting such a fault, an alternate configuration of the tile is activated which does not rely on the faulty CLB. Each implementation is interchangeable with the original, as the interface between the tile and the surrounding areas of the design is fixed and the tile's function remains unchanged. The timing of the circuit may vary, however, due to the changes in routing.

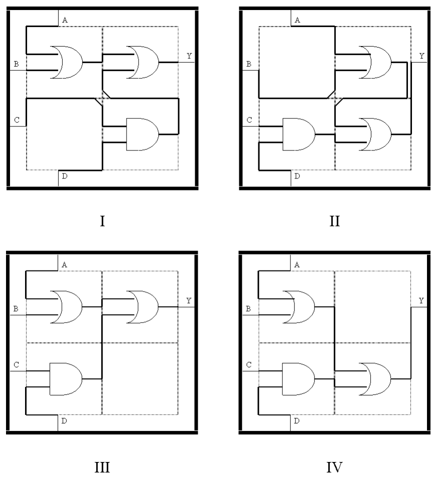

Figure 1: Motivational Example

This approach has three main benefits compared to redundancy-based fault-tolerance: very low overhead, the option for runtime management, and complete flexibility. The overhead required to implement this fine-grained approach, which can be measured in both physical resources on the FPGA (CLBs, I/O blocks, and routing) and timing, is extremely low compared to redundancy. Runtime management can be a very valuable feature of a system, particularly for mission-critical applications. This fault-tolerance 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-tolerance can be changed based on timing constraints, resource limitations, or presumed CLB reliability.

2.3 Paper Organization

The following two sections discuss background information, approach restrictions, and work related to fault-tolerance and FPGAs. Section 5 describes the details of the approach and implementation, and Section 6 introduces the formulas used to calculate the data for the experimental results in Section 7. Section 8 discusses future work on this topic, and the paper closes with some remarks summarizing the benefits of this fault-tolerance approach.

3. Preliminaries

In this section, we survey the relevant background material for the proposed approach. We present the targeted FPGA architecture, the fault model assumed, and techniques envisioned for supporting the testing and fault diagnosis steps of the approach.

3.1 FPGA Architecture Model

The new fault-tolerance approach is demonstrated using the Xilinx XC4000EX family as the target architecture, specifically the XC4028EXBG352 [24]. However, neither the general concept nor the optimization algorithms are specific to the 4000EX family, or even Xilinx architectures. Any FPGA architecture supporting the ability to reconfigure a large number of times could be used, such as the Altera 10k and the GateField flash memory devices. The approach is not applicable to anti-fuse systems, such as the Actel architecture, as they can not be reprogrammed.

3.2 Fault Model, Testing and Diagnosis

The proposed approach requires fault detection and a diagnosis method as a preprocessing step. We assume a widely used single stuck at, open, or short fault model [1]. It is interesting to note that our strategy actually covers many simultaneous faults as long as each tile (see Sections 5 and 6) has at most one faulty CLB. In its current form, our approach does not address interconnect faults. Note that for local interconnects, interconnect faults will be expressed as a fault of the CLB to which it connects.

A number of schemes have been developed for detecting faults in FPGAs through exhaustive testing of the device architecture. Most of these approaches can be classified as off-line. For example, with Built-In Self-Test (BIST) [13, 21, 8, 3], the FPGA is loaded with a small testing circuit that is restricted to a specific physical region of the device, which is then used to test another portion of the device. The test circuit is moved across the device in a systematic manner until the entire device is thoroughly tested. The downside of these approaches is that they require the device to be taken off-line, which may not be practical in highly fault-sensitive, mission-critical applications. Fault-detection latency also increases as a result of an off-line approach. Recently, an on-line testing scheme has been developed for bus-based FPGAs that avoids these problems and may be well suited for fault-detection within this fault-tolerance approach [18].

4. Related Work

Related work can be traced along the following three lines of research: FPGA synthesis, fault-tolerant design, and FPGA yield enhancement.

A number of different FPGA architectures and synthesis techniques have been proposed and demonstrated [16, 2]. Conceptually, our fault-tolerance approach is closest to BISR techniques. The main targets for BISR are systems that are bit-, byte-, or digit- sliced. These types of systems include SRAM and DRAM memories [14], as well as systems designed using a set of bit planes and arithmetic-logic units (ALUs), assembled from ALU byte slices [19]. By far the most important use of bit-sliced BISR is in SRAM and DRAM circuits [17, 9, 22]. The bit-sliced BISR in memories significantly increases memory production profitability and is regularly used in essentially all modern DRAM designs. Among other BISR bit-sliced devices, the most popular and well addressed, from both a theoretical and practical point of view, are programmable logic arrays PLAs [4, 10, 23, 6]. A simple, yet powerful methodology for the implementation of ALU byte slices was proposed by Levitt et. al. [11].

Howard et. al. [7] and Dutt et. al. [5] have proposed using similar regularly structured BISR techniques for improving FPGA yield. Spare resources are allocated, and a manufacturing step is used to swap spare CLBs for faulty components. Altera uses this approach, along with on-chip fuses, to increase production yield on the 10K parts. Mathur and Liu have proposed using modified place-and-route tools to reroute part of the net-list in the vicinity of a faulty CLB [12].

Our approach is completely transparent to the existing CAD tool chain and exists as an intermediate step that is used in conjunction with existing synthesis and place-and-route tools. Unlike the BISR techniques used in manufacturing, we are able to dynamically tolerate faults in the field. Finally, unlike Mathur and Liu, we are able to make timing guarantees (which is critical for real-time systems), require less system downtime, and do not require the end user to have access to FPGA CAD tools.

5. Approach

The key element of our approach to fault-tolerance is partially reconfiguring the FPGA to an alternate configuration in response to a fault. If the new configuration implements the same function as the original, while avoiding the faulty hardware block, the system can be restarted. The challenging step is to identify an alternate configuration efficiently. In this section, we elaborate on the key elements of our approach.

5.1 Tiles and Atomic Fault-Tolerant Blocks

We reduce the amount of configuration memory required by reducing the size of the component that is reconfigured. This is enabled by logically partitioning a design in a way that components can be independently reconfigured without impacting the rest of the design. In comparison with other alternatives, this approach also reduces the down time for devices that support partial reconfiguration and, more importantly, significantly increases the level of fault-tolerance with only nominal hardware and timing overhead. The key concepts for implementing the new approach are tiles and atomic fault-tolerant blocks (AFTBs).

Definition 1: A ***tile*** is composed of three elements: a set of CLBs and interconnect resources, a net-list which must be placed on those CLBs and routed across the interconnect, and a specification of how to interface the tile to adjacent tiles.

Definition 2: An ***atomic fault-tolerant block*** is one instance of a tile and has at least one spare CLB that serves to "cover" the faulty CLB(s).

Because each tile is associated with both physical resources and portions of the complete net-list, the design can only be partitioned into tiles after the complete net-list has gone through place-and-route once. By fixing the interfaces between the tiles (a constraint imposed on the place-and-route software), we create the opportunity to produce multiple partial configurations 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 CLB is detected, a configuration of the tile's functionality (i.e. an AFTB) that does not utilize the faulty CLB can be activated.

Each tile has a set of AFTBs. An AFTB is independent from all AFTBs associated with other tiles by virtue of the fixed tile interface. Thus, selecting one AFTB for each tile can assemble a complete configuration, under the condition that none of the AFTBs rely on a faulty component.

Tiling provides many advantages in the implementation of fault-tolerant FPGA systems. First, the amount of memory needed to store the set of AFTBs is smaller than the amount required to store a set of complete configurations. For example, consider a design that must be able to tolerate any single CLB fault and which maps into a 6x6 CLB array. It may be possible to divide the design into four 3x3 tiles (Figure 2). Assuming that one configuration of the complete 6x6 design requires *X* bytes of memory, the non-tiled approach would require *36*X* of memory for fault-tolerance: one configuration for each CLB that is at risk. With our method, each tile would require 9 AFTBs. However, since each tile (X/4 storage bytes) is independent, the entire storage is only *9*X*, a 75% reduction from the non-tiled approach.

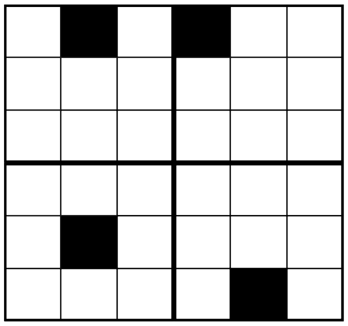

Figure 2: A 6x6 CLB design partitioned into 4 3x3 tiles

Tiling also increases reliability. For this example, the non-tiled approach could tolerate only one faulty CLB in the entire device. The tiled approach, however, is capable of tolerating any single fault in a tile but up to 4 faulty CLBs in the entire device.

The cost of increased fault-tolerance and reduced configuration memory is the possible introduction of more spare resources. For this example, the non-tiled approach reserves 2.7% of the CLBs 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 tool to the designer.

One remaining issue involves circuit timing. While timing analysis tools are not completely reliable for FPGA devices, the Xilinx XACT Step software has proven to be reasonably accurate. We use this tool to determine the timing estimate of the initial configuration before tiling. Furthermore, because each individual AFTB is generated as a modification to the original configuration we have good timing estimates for any single failure. However, it is difficult to know what the circuit timing will be once multiple AFTBs have been activated in response to failures in more than one tile. In particular, the critical path for the entire FPGA could pass through several AFTBs without falling on the longest path within any of them.

5.2 Synthesis Methods

The synthesis approach is organized in an iterative top-down manner. We start with a non-fault-tolerant base design and recursively partition it into tiles and AFTBs. We next check the feasibility of all fault scenarios in decreasing estimated level of difficulty. The idea is to terminate as early as possible those base designs that will not result in a feasible solution. We also calculate early the final reliability figures, so that the designer has an option of terminating the current search and starting one that has a higher reliability potential.

The synthesis is summarized in the following pseudo-code:

```
1.   while (!(complete || design possibilities exhausted)) {
2.       create initial non_ft_design;
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 AFTBs;
10.              calculate AFTB reliability;
11.              if (!meet reliability criteria) break;
12.              order tiles by ft realization difficulty;
13.              order AFTBs by ft realization difficulty;
14.              for (j=1; j<=# of tiles; decreasing difficulty) {
15.                  for (i=1; i<=# of AFTBs; decreasing difficulty) {
16.                      create ft_design(i,j);
17.                      if (!(success && meet timing criteria)) break;
18.  } } } } }
```

Lines 2-4 initialize the synthesis process for one instance of the base design. The place and route tool creates the base non-fault-tolerant design, and the relevant design characteristics are recorded. The procedure for the calculation of reliability is explained in Section 6.

Line 5 starts the synthesis algorithm and dictates that the loop will terminate upon the creation of a fault-tolerant design that meets all of the user specifications, including overhead (area and timing), level of fault-tolerance, and available memory. The loop will also terminate if the algorithm reaches the end of its exhaustive tile partitioning search, thus revealing that the specifications cannot be met for the given FPGA architecture and design generated in line 2. In this case, the complete algorithm will be repeated using a different base design with increased spare resources throughout the FPGA.

Line 6 partitions the design into tiles, as described in Section 5.1. The placement and shape of the tiles are determined by the following three key factors listed in decreasing order of importance: amount of interconnect across the tile interface, tile logic density, and tile size. Our reliability calculation (see Section 6) indicates that large tiles result in higher reliability. If the tiling attempt does not meet the user area specifications, the algorithm returns to the beginning of the tile partitioning loop for another tiling attempt. Hard macros (and fast carry chains) also affect the placement and shape of tiles, as efforts are made to keep macros intact.

Line 8 begins the AFTB partitioning algorithm, which terminates upon the successful creation of a fault-tolerant design meeting all user specifications or upon the exhaustion of all AFTB partitioning possibilities. If the latter occurs, the algorithm returns to line 5 for re-tiling.

Line 9 lays out the various AFTBs within a tile. The number of AFTBs for a given tile depends on the desired level of fault-tolerance, the number of free CLBs in the tile, and the malleability and density of the logic. The criteria used for partitioning the design into tiles are also used for this AFTB partitioning.

Line 10 insures that the tile and AFTB partitions meet the user reliability specifications, and line 11 returns the algorithm to the beginning of the AFTB partitioning loop if they are not met. If upon such a return the AFTB partitioning possibilities have been exhausted, the design must be re-tiled, returning the algorithm to line 5.

Lines 12 and 13 facilitate early synthesis process failure detection. Tiles and AFTBs that are less likely to successfully place and route should be attempted first, thus efficiently returning the algorithm to the beginning of the loop if a fault-tolerant design meeting user specifications is not possible with the current tile and/or AFTB partitioning. The tile and AFTB characteristics causing them to be more difficult to realize include the criteria used in tile and AFTB partitioning. The presence of macros, and therefore reduced logic malleability, may also impact the assigned order of realization difficulty.

Lines 14 and 15 enforce the order defined by the two previous steps, as line 16 attempts to configure the various tiles and AFTBs. If the design can not be configured or if the configuration doesn't meet user timing specifications, the algorithm returns to the beginning of the AFTB partitioning loop (line 8) or, if AFTB partitioning is exhausted, to the beginning of the entire synthesis algorithm for re-tiling (line 5). The next iteration of line 6 partitions the design giving more slack (i.e. free CLBs) to the area of the previous iteration's failing tile. If no other tiling possibilities exist, the algorithm returns to line 1 to generate a new base design. If no base design possibilities remain, the algorithm terminates as unsuccessful.

The synthesis approach is illustrated using the PREP 5 benchmark shown in Figures 3-5. Figure 3 shows an implementation of the PREP 5 benchmark on the Xilinx 4000 architecture, a configuration that occupies rows 18-24 and columns 1-4. Only the CLBs in the defined area are used in the tiled design; the remaining CLBs are prohibited from use in any of the AFTBs.

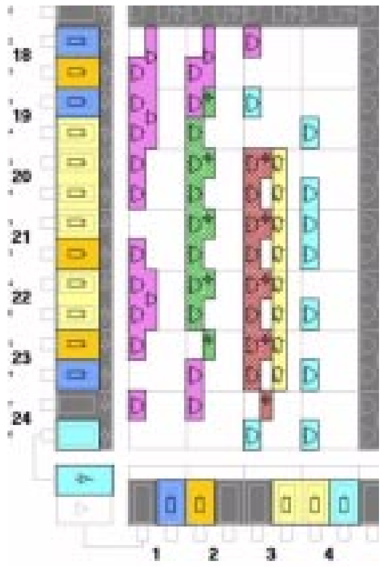

Figure 3: Initial floorplan for PREP 5 benchmark

Tile A covers rows 18-24 and columns 1-2, and tile B covers rows 18-24 and columns 3-4. Tiling was restricted by the occurrence of hard macros in each tile.

After partitioning the design into a set of tiles, the tiles are ordered by their implementation difficulty. In Figure 3, the tiles are ordered A first and B second, primarily because the logic in tile A is denser.

Next we create a set of AFTBs for each tile, which are sorted in decreasing order of implementation difficulty. The tiles in Figure 3 are assigned AFTBs that each possess two adjacent free CLBs. Each CLB is covered only once, making the total number of AFTBs per tile seven. For each AFTB, a complete configuration (i.e. one AFTB for each tile) is passed through the place and route tool. Although the placement and routing of the logic in the tile can be quite different for each AFTB, variations within a tile do not propagate to other tiles (other than timing). This is possible because, for each fault detected, only the tile in question is changed. All other tiles remain the same as in the original configuration.

Figure 4 shows the AFTB that was attempted first for the design in Figure 3.

5.3 Enforcing Fault-Tolerance at Run-time

After all of the AFTBs are stored in memory and the circuit begins operation, the system runs normally with the original configuration until a fault is detected. Upon detection, the circuit ceases functional mode until the proper reconfiguration can be made. As already mentioned, we assume that the fault detection system is able to identify the faulty resource in an architecture map. This information allows the system to retrieve the appropriate AFTBs from the configuration memory. The time needed for this memory access depends on normal access factors: access size, memory bus width, memory size, etc.

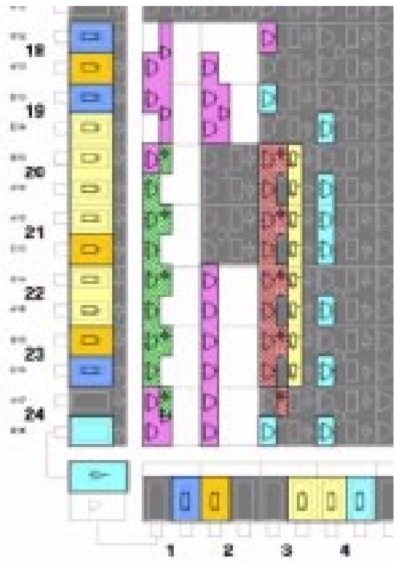

Figure 4: PREP benchmark 5 after tiling with one AFTB

The last step involves the actual reconfiguration of the FPGA device. Once the AFTB is retrieved from memory, two options are possible depending on the capabilities of the FPGA architecture. If the device supports partial reconfiguration, the AFTBs of the affected tiles can be used in isolation to directly update the configuration. Otherwise, the AFTBs must be merged with the active and functioning AFTBs, thus providing the necessary data for a total chip reconfiguration. For example, if the CLB at row 20, column 3 failed in the design of Figure 3, the proper configuration for tile B would be fetched from memory and configured onto the FPGA. The result is shown in Figure 5. Note that the interface between tiles A and B is unchanged.

The time required to update a tile with a new AFTB depends on the complete set of tiles, the device architecture and the surrounding computer system. However, in all cases, it is bounded and can be used to provide a measured level of system availability. After updating the affected tile, the device is reset and the system resumes operation as before the fault. The only possible change could be in the timing of the circuit, as the routing in the altered tiles has likely changed. Timing numbers are generated with each AFTB, and the system can operate under worst case assumptions. Another, technically more demanding, option is the use of a programmable clock. If new faults occur later, the process repeats itself until more than one fault occurs in a tile for which there is no available AFTB that has all faulty CLBs as unused.

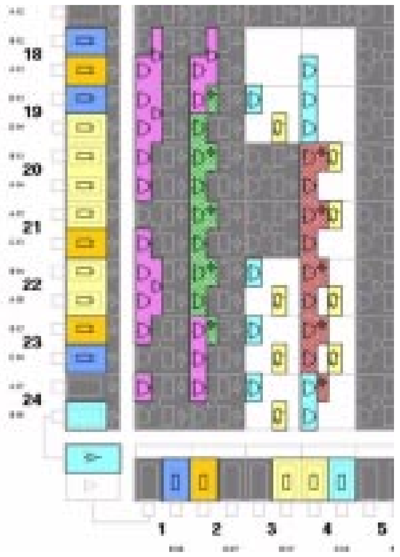

Figure 5: System at runtime after swapping the AFTB in tile B due to fault at (20,3)

6. Reliability Calculation

The first step in calculating reliability is the selection of fault models. There are two major sources of logic faults in FPGA systems: cosmic radiation and manufacturing/ operating imperfections.

Since the size of radiation particles is usually small when compared to the size of modern FPGA CLBs, we selected a cosmic radiation fault model that follows uniform distribution of independent (non-correlated) failures. Extensive terrestrial efforts to accurately model the rate of such soft faults indicate high variance (several orders of magnitude) depending on factors such seasonal solar activity, altitude, latitude, device technology, and device materials. Even for the same chip from the same manufacturer, variations by a factor higher than 200 are not uncommon [25]. Experiments indicate that in FPGA-like devices at an altitude of 20 km, error rates significantly higher than once per 1000 hours are common [15]. Also, as circuit devices become smaller, they become more sensitive to soft faults [25]. If one considers the multiyear life of computing devices and other sources of potential errors (e.g. power surges), the need for fault tolerance in devices which implement critical functions becomes apparent.

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 the design). For this type of model, we follow the gamma-distribution Stapper fault model [20]. The model is applicable on any integrated circuit with regular repetitive structure, including memories and FPGA devices.

In the remainder of the section, we elaborate on technical details related to the two fault models' reliability calculations.

6.1 Independent Uniformly Distributed Faults

Suppose that a design is partitioned into t tiles. Furthermore, assume that tile i has a total of c_i AFTBs. The total number of used CLBs in the initial design is t_n. For the sake of clarity and simplicity, we limit our discussion to the case where each AFTB consists of three or fewer CLBs. We denote by m_{1i}, m_{2i}, m_{3i} the number of AFTBs of size one, two, and three CLBs respectively in tile i. We also denote their weighted sum $m_i = m_{1i} + 2 * m_{2i} + 3 * m_{3i}$.

Finally, we assume that the probability of a CLB being faulty is $(1-P)$ (i.e. the probability that a CLB is fault free is P). It is easy to see that the probability, P_{init}, that the original design is fault free is $P_{init} = P^{tn}$.

It is also easy to verify that the probability that a tile i in the optimized design is fault free, Pft_i, is given by the following formula:

$$Pft_i = P^{m_i} + m_i * P^{(m_i-1)}(1-P) + (m_{2i} + 3 * m_{3i}) * P^{(m_i-2)} * (1-P)^2 + m_{3i} P^{(m_i-3)} * (1-P)^3$$

The first term corresponds to the scenario where all CLBs are fault free. The last three terms correspond to the scenarios where one, two, and three CLBs are faulty, respectively. The probability (Pft) that the optimized fault-tolerant design is functional is $Pft = \prod_{i=1}^{t} Pft_i$.

6.2 Stapper's Fault Model

To calculate reliability for correlated faults, we started from the following formula [20]:

$$\overline{Y}_{mn} = \binom{n}{m} \overline{Y}_1^{m} \left(\prod_{i=0}^{m-1} \frac{\mu + i}{\mu + i\overline{Y}_1} \right) \times \left(1 - \overline{Y}_1\right)^{n-m} \left(\prod_{j=0}^{n-m-1} \frac{\mu + j\overline{Y}_1 / \left(1 - \overline{Y}_1\right)}{\mu + m\overline{Y}_1 + j\overline{Y}_1} \right)$$

(Note that we have fixed a small typographical error in the original published formula.)

The formula calculates the probability that exactly *m* out of *n* identical modules operate correctly for a given value of the variability parameter μ and single CLB reliability Y_1. The parameter μ indicates the assumed or the measured probability of clustered faults. Small values of μ imply high levels of clustering. As μ tends toward infinity the formula reduces to the case of independent uniformly distributed faults.

Stapper's formula is used to calculate the probability that at least *m* out of *n* modules (in this case CLBs) operate correctly by a direct summation of relevant terms. This helps reveal how effective this fault-tolerance approach is in the face of several clustered CLB faults.

7. Experimental Results

We conducted an evaluation of the proposed approach and optimization algorithms in two phases. In the first, we applied the approach to nine MCNC designs. In the second phase we studied expected reliability improvement trends as a function of the number of used CLBs in a design.

Tables 1 and 2 show timing and cost (area) metrics respectively of the designs before and after the application of the new approach for reliability enhancement. The first column in both tables indicates the name of a design. The next four columns in Table 1 show the initial delay, and the best, worst, and median delay of the optimized designs. The rightmost column indicates the timing overhead as a result of enhanced reliability. For all nine designs, the largest timing overhead was in the range of 14% to 45%.

A number of factors complicate the task of calculating the physical resource overhead. The place-and-route tools indicate the number of CLBs that are used for a particular placement. However, these utilized CLBs rarely are packed into a minimal area. Unused CLBs introduce flexibility into the place-and-route step that may be essential for completion or good performance. For example, the initial c880 design has a concave region that contains 42 utilized CLBs but also 10 unutilized CLBs (19%). Therefore, we will report overhead in terms of the area used by the fault-tolerant design minus the total area of the original design, including unused CLBs such as the 19% measure above. The area overhead is presented in Table 2, using the same format as Table 1. The average, median and worst-case area overheads were 5.4%, 5.3%, and 9.8% respectively.

Table 3 shows reliability improvements for the MCNC benchmarks under the uniform random fault model. The first column indicates the assumed probability (p) that a CLB is fault free. The next two columns show the probability that the original and fault-tolerant design of a particular benchmark is functioning properly. For example, for 9sym, with p = 0.995, the probability of the initial design and tiled design being functional is 81.0% and 98.4% respectively.

Table 4 shows the reliability figures for the same set of designs (original and tiled) with four different variability factors, μ, assuming the probability that a CLB is fault free is 90% and 99%. Table 5 is, in a sense, the strongest indication of the effectiveness of the proposed approach for

Design	Initial (ns)	Fastest (ns)	Slowest (ns)	Median (ns)	Slowest – Fastest
					Fastest
9sym	71.6	71.6	82.0	76.8	0.15
c499	104.9	104.9	130.0	113.6	0.24
c880	110.8	110.8	126.4	117.3	0.14
duke2	87.9	87.9	118.8	96.4	0.35
rd84	50.2	50.2	72.8	58.6	0.45
planet1	145.0	145.0	194.9	166.1	0.34
styr	150.6	150.6	189.8	167.2	0.26
s9234	135.0	135.0	183.6	153.2	0.36
sand	97.6	97.6	117.7	103.8	0.21

Table 1: Timing bounds due to routing variation among AFTBs for each tile

Design	Original # of CLBs	Final # of CLBs	Final - Original
			Original
9sym	46	49	.065
c499	94	96	.021
c880	110	115	.045
duke2	93	100	.075
rd84	27	28	.037
planet1	95	100	.053
styr	78	81	.038
s9234	195	206	.056
sand	82	90	.098

Table 2: Variation of resources used among AFTBs for each tile

CLB P	.900		.950		.990		.995		.998		.999		.9999	
Design	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled
9sym	1.2	16.9	11.6	56.2	65.6	95.7	81.0	98.4	91.9	99.5	95.9	100	99.2	100
c499	0.01	1.63	3.2	37.5	43.0	89.0	65.6	95.6	84.5	98.6	91.0	99.3	99.2	100
c880	0.0	0.6	1.8	31.7	37.3	91.2	61.2	97.6	82.2	99.6	90.7	99.9	99.0	100
duke2	0.01	0.7	2.8	31.9	41.3	90.8	64.3	97.5	83.8	99.6	91.6	99.9	99.1	100
rd84	7.2	38.6	27.7	74.7	77.8	98.5	88.2	99.6	95.1	99.9	97.5	100	99.8	100
planet1	0.02	5.4	11.5	32.6	41.7	94.1	64.7	98.4	84.0	99.7	91.7	99.9	99.1	100
styr	0.01	2.9	2.5	31.4	48.5	93.8	69.7	98.3	86.6	99.7	93.0	99.9	99.2	100
s9234	0.0	0.003	0.01	3.17	16.1	82.0	40.2	94.8	69.5	99.1	83.3	99.8	98.2	100
sand	0.03	1.53	1.83	2.50	45.7	92.4	67.6	97.9	85.5	99.7	92.5	99.9	99.2	100

Table 3: Reliability of the original vs. tiled designs against CLB reliability

	1		2		5		20	
	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled
9sym	62.6/95.8	72.0/97.2	48.5/93.5	62.8/96.4	28.5/89.0	40.1/94.9	8.77/79.6	29.7/94.1
c499	57.9/95.1	68.2/96.5	41.6/92.2	54.8/95.5	19.9/86.1	36.2/94.1	2.76/71.6	14.5/88.7
c880	55.9/94.9	65.7/96.3	40.2/91.9	53.2/95.2	18.3/85.4	33.8/93.3	2.08/69.8	11.2/87.2
duke2	57.6/95.0	67.9/96.3	41.1/92.1	54.2/95.4	19.4/85.9	25.9/93.9	2.54/71.0	14.3/88.4
rd84	66.4/96.3	76.7/97.7	54.3/94.5	70.1/96.3	36.9/91.1	56.4/95.6	18.0/85.1	52.8/95.1
planet1	57.7/95.0	67.9/96.3	41.3/92.2	54.2/95.4	19.5/86.0	25.9/93.9	2.59/71.2	14.3/88.4
styr	58.9/95.2	68.1/96.9	43.0/92.5	56.2/95.8	21.6/86.8	39.6/94.4	3.63/73.4	15.4/89.8
s9234	53.1/94,3	64.4/95.6	35.1/90.9	58.9/93.5	13.1/82.9	28.4/89.6	0.63/62.5	5.32/83.6
sand	58.4/95.1	68.0/96.6	42.3/92.3	55.3/95.6	20.7/86.4	32.1/94.2	3.15/72.3	14.8/89.2

Table 4: Reliability of original and tiled designs using Stapper's correlated failure model with CLB reliability of 90%/99%

	CLB Overhead for Tiling	Random Fault Model		Stapper's Fault Model	
		Orig.	Tiled	Orig.	Tiled
9sym	6.5%	2.4	16.9	16.8	29.7
c499	2.1%	0.02	1.6	5.4	14.5
c880	4.5%	0.00	0.6	4.1	11.2
duke2	7.5%	0.02	0.7	5.0	14.3
rd84	3.7%	13.9	38.6	32.8	52.8
planet1	5.3%	0.04	5.4	5.1	14.3
styr	3.8%	0.02	2.9	7.1	15.4
s9234	5.6%	0.00	0.003	1.3	5.32
sand	9.8%	0.06	1.5	6.2	14.8

Table 5: Comparison of reliability and overhead for original design with complete redundancy (i.e. 100% overhead) vs. tiled design for CLB reliability of 90% and μ = 20

CLB	100 CLB design		1000 CLB design		5000 CLB design	
Reliability	Orig.	Tiled	Orig.	Tiled	Orig.	Tiled
.9500	0.005921	0.444669	0.000000	0.000302	0.000000	0.000000
.9750	0.079551	0.800119	0.000000	0.107534	0.000000	0.000014
.9800	0.132687	0.864375	0.000000	0.232820	0.000000	0.000684
.9850	0.220739	0.919633	0.000000	0.432660	0.000000	0.015161
.9900	0.366277	0.962643	0.000043	0.683364	0.000000	0.149026
.9950	0.606224	0.990317	0.006704	0.907280	0.000000	0.614762
.9980	0.819220	0.998429	0.136145	0.984404	0.000047	0.924414
.9990	0.905528	0.999608	0.370696	0.996091	0.007000	0.980610
.9995	0.951999	0.999903	0.611453	0.999028	0.085470	0.995153
.9999	0.990868	0.999995	0.912346	0.999952	0.632119	0.999762

Table 6: Reliability of traditional design methods vs. tiled approach against CLB reliability for large FPGAs

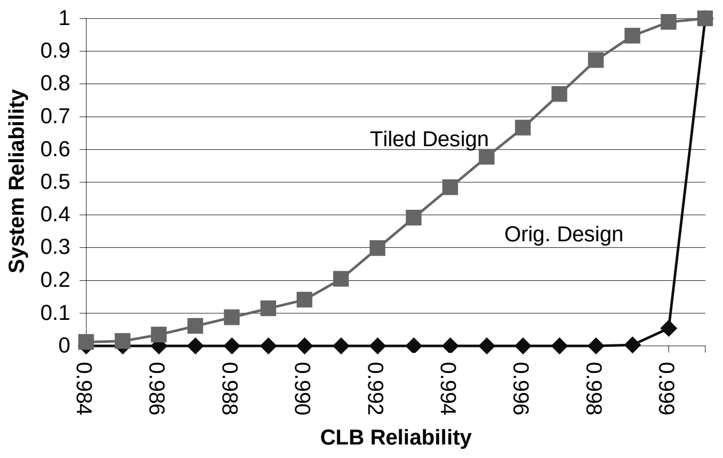

Figure 6: Reliability of traditional methods vs. tiled methods for a hypothetical 5000 CLB FPGA

reliability improvement. The second column of this table indicates the area overhead of tiled designs. The next four columns provide reliability data under two selected fault models for the duplication-enhanced fault-tolerant original and for the tiled designs. For both models, the tiled designs have significantly lower area overhead and always higher reliability than the conventional fault-tolerant designs.

Finally, Table 6 calculates reliability improvement trends as the size of designs increase. It is assumed that all designs are partitioned into tiles of 5 AFTBs each consisting of two CLBs and requires an average hardware overhead (i.e. ~5.4%). Table 6 indicates the potential of the proposed approach for reliability enhancement. For example, in the case of a 5000 CLB design, with $p = 0.999$, the probability of the initial design being functional is less than 1%, while the probability of the tiled design being functional is 98%. Figure 6 graphs the reliability results for the 5000 CLB design.

8. Future work

Many of the highest volume FPGA devices tend to be dominated by interconnect resources, e.g. the Xilinx 4000 and the Altera 10K families. On the Xilinx 4000EX series, the majority of configuration bits are used to program the state of the interconnect rather than the CLBs, and it is likely that these interconnect resources are more susceptible to faults. The fault-tolerance methodology presented above addresses faults in interconnect resources directly dedicated to specific CLBs because they appear as CLB faults. Unfortunately, the vast majority of interconnect resources pass through higher-level hierarchical switch structures that are not covered by unique CLB faults. Some of these routing resources will remain unused in each AFTB, thus providing some additional fault-tolerance. However, since this benefit comes as a byproduct of the approach rather than as a primary goal, we currently cannot make any specific claims on interconnect fault tolerance.

9. Conclusions

Fault-tolerant techniques have recently emerged as an important design consideration for FPGA-based systems due to the rapid progress in FPGA integration and the growing market for these devices. In order to address this problem, we have developed the first fault-tolerance approach to work at the level of physical design. Our hierarchical fault-tolerance technique partitions designs into tiles and atomic fault-tolerant blocks. The approach scales systematically through an exploration of the design solution space at the physical level. The approach is constructed of four phases: design partitioning, tile partitioning and ordering, AFTB partitioning and ordering, and reliability calculation.

Experimental results conducted on a subset of the MCNC benchmarks for large CLB FPGAs indicate that the technique is effective with low hardware overhead.

10. Acknowledgments

The authors would like to thank Prof. Jason Cong, John Peck, Hea Joung Kim, and Jason Leonard for their assistance. This work was supported by the Defense Advanced Research Projects Agency of the United States of America, under contract DAB763-95-C-0102 and subcontract QS5200 from Sanders, a Lockheed Martin company.

11. References

[1] Abramovici, M., et. al. *Digital Systems Testing and Testable Designs*, New York, Computer Science Press, 1990.

[2] Carter, W. S., et. al. “A User Programmable Reconfigurable Logic Array”, *Proceedings of the Custom Integrated Circuits Conference*, pp. 233-235, 1986.

[3] Chen, X. T., et. al. “A Row-Based FPGA for Single

and Multiple Stuck-At Fault Detection," *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, 1995.

[4] Greene, J. W. and A. E. Gamal, "Configuration of VLSI Arrays in the Presence of Defects", *Journal of the ACM*, vol. 31, no. 4, pp. 694-717, 1984.

[5] Hanchek, F. and S. Dutt, "Node-Covering Based Defect and Fault-Tolerance methods for Increased Yield in FPGAs", *Proceedings of the Ninth International Conference on VLSI Design*, pp. 225-229, 1995.

[6] Hassan, N. and C. L. Liu, "Fault Covers in Reconfigurable PLA's", *Proceedings of the International Conference on Fault-Tolerant Computing*, pp. 166-173, 1990.

[7] Howard, N. J., et. al. "The Yield Enhancement of Field-Programmable Gate Arrays", *IEEE Transactions on VLSI Systems*, vol. 2, pp. 115-123, 1994.

[8] Huang, W. K. and F. Lombardi, "An Approach for Testing Programmable/Configurable Field Programmable Gate Arrays," *IEEE VLSI Test Symposium*, 1996.

[9] Kikuda, S., "Optimized Redundancy Selection Based on Failure-Related Yield Model for 64-Mb DRAM and Beyond", *IEEE Journal of Solid State Circuits*, vol. 26, no 11, pp. 1550-1555, 1991.

[10] Koren, I. and D. K. Pradhan, "Introducing Redundancy into VLSI Designs for Yield and Performance Enhancement", *International Conference on Fault-Tolerant Computing*, pp. 330-335, 1985.

[11] Levitt, K. N., et. al. "A Study of the Data Communication Problems in Self-Repairable Multiprocessors", *Conference Proceedings of AFIPS*, Washington, D. C., Thompson Book, pp. 515-527, 1968.

[12] Mathur, A. and C. L. Liu, "Timing Driven Placement Reconfiguration for Fault-Tolerance and Yield Enhancement in FPGAs", *Proceedings of the ED&TC 96*, pp. 165-169, 1996.

[13] Michinishi, H., et. al. "A Test Methodology for Configurable Logic Blocks of a Look-up Table Based FPGA," *Transactions of the Institute of Electronics, Information and Communication Engineers*, vol. J79D-I, pp. 1141-1150, 1996.

[14] Moore, W. R., "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield", I, pp. 684-698, 1986.

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

[16] Rose, J., et. al. "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency", IEEE *Journal of Solid State Circuits*, vol. 25, pp. 1217-1225, 1990.

[17] Sarrazin, D. B. and M. Malek, "Fault-Tolerant Semiconductor Memories", *IEEE Computer*, vol. 17, no. 8, pp. 49-56, 1984.

[18] Shnidman, N., W. H. Mangione-Smith, and M. Potkonjak, "Fault Scanner for Reconfigurable Logic," *Advanced Research in VLSI*, Ann Arbor, MI, 1997.

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

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

[21] Stroud, C., et. al. "Built-In Self-Test of Logic Blocks in FPGAs (Finally, a Free Lunch: BIST Without Overhead!)," *IEEE VLSI Test Symposium*, 1996.

[22] Tanabe, A., et. al. "A 30-ns 64-Mb DRAM with Built-in-Self-Test and Self-Repair Functions", *IEEE Journal of Solid State Circuits*, vol. 27, no. 11, pp. 1525-1533, 1992.

[23] Wey, C. L., et. al. "On the Design of a Redundant Programmable Logic Array (RPLA)", *IEEE Journal of Solid-State Circuits*, vol. 22, no. 1, pp. 114-117, 1987.

[24] Xilinx, *The Programmable Logic Data Book*, San Jose, CA: 1996.

[25] Ziegler, J. F., 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.