Fast Run-Time Fault Location in Dependable FPGAs

Wei-Je Huang, Subhasish Mitra, and Edward J. McCluskey

TR 01-5 May 2001	**Center for Reliable Computing** Gates Building 2A, Room 236 Computer Systems Laboratory Dept. of Electrical Engineering and Computer Science Stanford University Stanford, California 94305

Abstract:

Run-time fault location in Field-Programmable Gate Arrays (FPGAs) is important because the resulting diagnostic information is used to reconfigure the FPGA for tolerating permanent faults. In order to minimize the system downtime and increase availability, a fault location technique with very short diagnostic latency is desired. In this paper, we present a fast approach for run-time FPGA fault location that can be used for high-availability reconfigurable systems. By integrating FPGA fault tolerance and Concurrent Error Detection (CED) techniques, our approach can achieve significant availability improvement by minimizing the number of reconfigurations required for FPGA fault location and recovery. The area overhead of our approach is studied and illustrated using applications implemented in FPGAs.

Funding:

This research was supported by the Advanced Research Projects Agency under Contract No. DABT63-97-C-0024.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. Introduction

The abundance of programmable logic and routing resources in current-generation Field-Programmable Gate Arrays (FPGAs) makes it possible to optimize an FPGA-based system for various perspectives, including performance, cost, and dependability. From the performance perspective, parallel architectures for various applications can be implemented in FPGAs to increase throughput [Hauck 98, Huang 00a]. From the cost perspective, optimized designs for different applications can be time-multiplexed in the same FPGA to improve hardware efficiency [Trimberger 98].

For FPGA-based reconfigurable systems used in dependable applications, the FPGA can be reconfigured to operate without using the defective element. In this way, the *Field Replaceable Unit* (FRU) is a fine-grained programmable logic block or a routing resource, instead of a chip or a board as in conventional fault-tolerant systems. Hence, unlike conventional fault-tolerant systems, FPGA-based reconfigurable systems do not need additional chips or boards as standby spares. The fine granularity of FRUs in FPGAs provides a more cost-effective solution to tolerance of permanent faults in the system.

Techniques for developing an FPGA-based dependable system include *Concurrent Error Detection* (CED) schemes for detecting errors during system operations, transient error recovery, tolerance for permanent faults, and fault location. Various CED techniques have been presented and analyzed to guarantee data integrity [Mitra 00a, Saxena 00, Touba 97, Zeng 99]. By *data integrity*, we mean that the system either produces correct outputs or indicates errors when incorrect outputs are produced. Correctness of the system is verified by redundant computations either in the time domain or using extra hardware. Redundant computations for error detection include the prediction of some special characteristics of the system output, and a *checker* that compares such characteristics obtained from the system output and the prediction. Output characteristics that are used for error detection can be the output itself, its parity, 1's or 0's count, transition count, etc. CED schemes typically have short latency of detecting errors.

Transient error recovery in FPGAs has been studied in two aspects. Traditional system-level approaches, such as rollback or roll-forward techniques [Pradhan 96, Huang 00b], can be used for restoring normal operations from transients that do not change the FPGA configuration data. On the other hand, configuration readback and writeback operations can be used to detect and correct transient errors in FPGA configuration memory [Carmichael 99, Huang 01a].

To tolerate permanent faults, FPGAs can be reconfigured so that the faulty parts are avoided in the new configuration. Previous research can be broadly classified into two categories: (1) fast re-mapping and rerouting techniques for dynamic run-time generation of alternative configurations after error detection and fault location [Emmert 98, Hanchek 98, Mahapatra 99, Lakamraju 00], and (2) precompiled configuration approaches that create alternative configurations during the design phase and load the appropriate configuration after error detection and fault location [Lach 98, Huang 01b].

Run-time fault location provides information about parts to avoid in FPGAs before a new, fault-tolerant configuration is loaded. Previous work on FPGA fault location can be found in [Jordan 93, Lombardi 96, Stroud 97, Stroud 98, Renovell 98, Mitra 98, Das 99, Abramovici 99]. However, these techniques generally require tens or hundreds of reconfigurations to diagnose the fault location, which results in a major component of the system downtime for FPGA-based reconfigurable systems. Therefore, there is a need for fast FPGA fault location that reduces the number of reconfigurations for diagnosis and integrates well with the subsequent FPGA fault tolerance scheme for reducing the overall downtime.

In this paper, we present a fast, run-time FPGA fault location approach. Our approach is based on the column-based precompiled configuration schemes for FPGA fault tolerance [Huang 01b], where alternative configurations for the application circuitry in the FPGA are precompiled with certain columns being intentionally unused. Instead of designing special configurations for fault location, we use the existing CED schemes in the application circuitry to find an alternative configuration that avoids the fault for the target application circuitry. If CED checkers are distributed in each sub-circuit of the system, where each sub-circuit is isolated by flip-flops, we can localize suspect faulty columns and reduce the number of configuration attempts. Moreover, using a modified column-based precompiled configuration scheme that generates alternative configurations by shifting sub-circuits, the enhanced version of our technique further reduces the number of reconfigurations for recovering from a failure. Availability of the FPGA-based system is thus improved significantly.

The organization of this paper is as follows. In Sec. 2, we briefly describe the FPGA model used in this paper. In Sec. 3, we discuss previous work related to this paper, including the dependable FPGA-based system architecture, previous FPGA fault location techniques, and the concept of a column-based precompiled configuration scheme for FPGA fault tolerance. In Sec.

4, we present the baseline version, blind configuration attempts, of the proposed fault location approach. In Sec. 5, we present an enhanced version of the baseline approach, which minimizes the number of configuration attempts. Area overhead in the enhanced approach is also discussed and illustrated by an example implemented in FPGAs. Section 6 concludes the paper.

2. FPGA Model

Figure 2.1 shows the model of the programmable logic core of the FPGA used in this paper. In this model, the programmable logic core of the FPGA consists of an array of three basic elements: *Configurable Logic Blocks* (CLBs), *Connection Boxes* (CBs), and *Switch Boxes* (SBs).

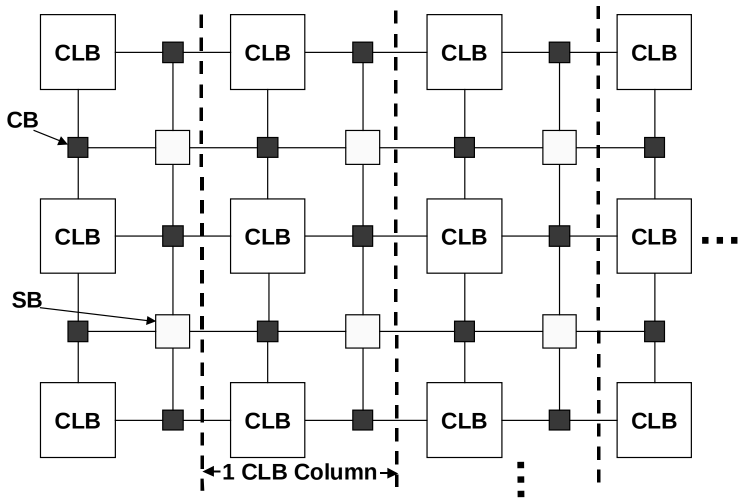

Figure 2.1: Architecture of the programmable logic core in FPGAs.

A CLB is the basic building block in the two-dimensional programmable logic core of an FPGA. It contains several SRAM lookup tables (LUTs) used to store user-defined combinational logic functions. It also contains flip-flops, multiplexers, and dedicated circuitry for optimizing the performance of user applications. CLBs are connected through horizontal and vertical wiring channels between neighboring rows and columns.

Current-generation FPGAs have various lengths of wires for connecting CLBs that are separated by different numbers of blocks. For example, in Xilinx Virtex-series FPGAs, *single lines* connect adjacent CLBs, while *hex lines* connect CLBs that are three or six blocks apart [Xilinx 01]. Signals on the wires are directed among CLBs and wiring channels by two types of routing matrices, CBs and SBs. CBs connect the inputs and outputs of a CLB to the adjacent wiring channels. SBs route horizontal and vertical wiring channels other than the I/Os of CLBs. Both CBs and SBs are matrices of Programmable Interconnect Points (PIPs). The states of the PIPs in these switch matrices are controlled by SRAM cells, which are configured according to the desired functionality.

In this architecture, a *CLB column* includes all CLBs and the corresponding switch matrices (CBs and SBs) in the same column of the array. The actual circuitry, programmable logic and routing resources, and the configuration architecture of each CLB column are identical. A typical example of the FPGA architecture used in this paper is the Xilinx Virtex-series FPGAs [Xilinx 01].

3. Related Work

3.1 High-level Architecture of FPGA-based Dependable Systems

High-level architectures for building dependable FPGA-based systems can be found in [Saxena 00] and [Mitra 00b]. For example, Fig. 3.1 shows a dependable, *dual-FPGA* architecture in [Mitra 00b]. In this example, each FPGA is configured to run certain applications with some CED schemes. Error signals reported by CED schemes are observed by a controller configured in the other FPGA. When the controller observes the error signals from the other FPGA, it initiates subsequent fault location and recovery operations for repairing the other FPGA.

Note that the controllers that carry fault location and recovery operations should be simple in order to avoid large area overhead. Therefore, run-time fault location and recovery techniques should also be as simple as possible to be feasible for FPGA-based dependable systems.

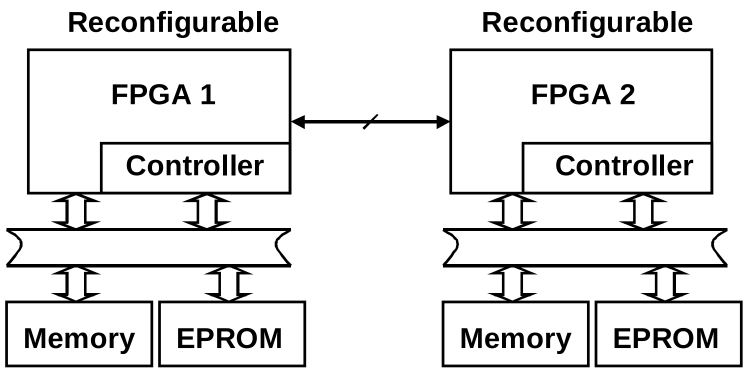

Figure 3.1: Dependable dual-FPGA architecture.

3.2 Previous Fault Location Techniques

Most of the papers on FPGA fault detection and location [Jordan 93, Lombardi 96, Stroud 97, Stroud 98, Renovell 98] focus on production test of FPGAs. Application dependent fault-location approaches have been described in [Mitra 98, Das 99]. While the diagnostic resolution of these approaches is very fine-grained (e.g., faulty CLBs), the number of reconfigurations (and hence, the fault-location time) can be large (unless partial reconfiguration capabilities based on selective reprogramming of CLBs are available).

The roving STAR approach described in [Abramovici 99, Emmert 00] mainly targets fault detection and location during system operation for fault-tolerant applications. In the roving

STAR approach, two rows and two columns of the FPGA are reserved as the self-testing area (STAR) for detection of permanent faults. The STAR areas are tested using special diagnostic configurations while the rest of the system is in operation. After the testing of a STAR is complete, the STAR moves to a new location until the entire FPGA is completely tested. Complete testing of a STAR in a Lucent ORCA 2C15A FPGA [Lucent 01] takes about 31 seconds, and the system clock has to be stopped for about 3 to 4 seconds for copying the state before the STAR moves to a new location [Emmert 00].

One concern with the roving STAR approach is the possible performance and availability degradation of the system because of the moving of the STAR. For example, if the clock frequency is 100 MHz, there will be a loss of 300 million cycles every 3,100 million cycles of operation of the system – since the system is unavailable during these 3 to 4 seconds, this implies degradation of system availability even when failures are not present in the system. Another cause of concern about the roving STAR technique is the error latency. It has been reported in [Emmert 00] that it takes approximately 6 minutes before the STAR has roved over the entire chip for an FPGA with 20 columns. Hence, unless CED techniques are used, it is possible that for an FPGA with 20 columns, it may take 6 minutes (approximately 36,000 million cycles for a 100 MHz system) before a fault is diagnosed. The above problems can be reduced if fine-grained CED techniques are used and, depending on the responses of the checkers, fault-location techniques (possibly using a STAR approach) are used. Using the STAR approach without utilizing the checker data can be very expensive in terms of the fault-location time and hence, system down-time.

3.3 Column-Based Precompiled Configuration Scheme

The choice of run-time fault location techniques for FPGA applications is closely related to the reconfiguration scheme used for tolerating permanent faults in the system. For example, if the dynamic re-mapping technique in [Emmert 98] or the tile-based precompiled configuration scheme in [Lach 98] is used for tolerating faulty CLBs, a fault location technique with diagnostic resolution of one CLB is required. Alternatively, if the localized swapping technique in [Lakamraju 00] is used, it is necessary to locate the faulty resource within a CLB.

In this paper, we present a fast, run-time FPGA fault location approach based on the column-based precompiled configuration techniques developed in [Huang 01b] for tolerating permanent faults. Column-based precompiled configuration techniques achieve fast reconfiguration with small storage overhead for the precompiled, alternative configurations.

The basic concept of the precompiled configuration approach is to generate alternative versions of logic placement and routing information of the same application in the FPGA during the design phase. In each configuration version, a certain part of the FPGA is intentionally unused. In this way, each configuration version can tolerate permanent faults in unused resources. Because alternative configurations are pre-generated during the design phase, the post-fault-location downtime is minimized.

In the column-based approach, each alternative configuration is created by shifting part or the entire of the original configuration in units of columns. Figure 3.1 shows an example of the *overlapping* column-based precompiled configuration scheme. Because the corresponding columns in which certain functions are mapped in different configuration versions are highly similar, using run-length coding on the configuration difference between such columns significantly reduces the storage overhead for alternative configurations. For a *k*-column circuit with *m*-column fault tolerance, the total number of alternative configurations required is ($C(k+m, m) - 1$) for the overlapping precompiled configuration scheme.

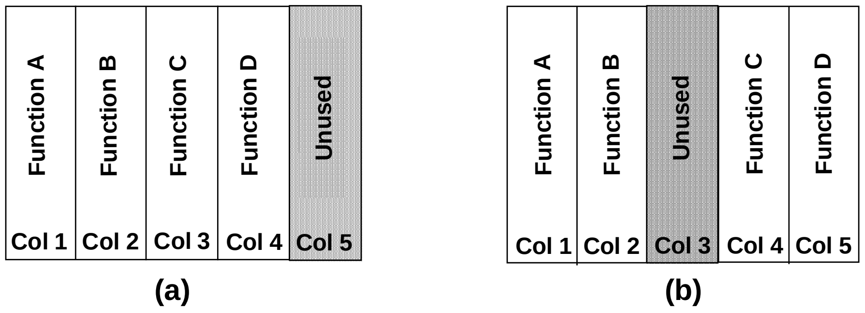

Figure 3.1: The overlapping precompiled configuration. (a) Original configuration. (b) Alternative configuration when column 3 is intentionally unused.

4. The Baseline Fault Location Scheme: Blind Configuration Attempts

When alternative configurations are available in the system, one simple way to locate the fault while repairing the system is to try all possible configurations alternately until a configuration that successfully avoids the fault is loaded. For example, successful configurations in the column-based precompiled configuration scheme can be the configurations that the faulty column is intentionally unused, or the configurations that the faulty block is happened to be empty because of the shifting of columns. For each configuration, CED schemes for the application circuitry are used to determine if the attempt is successful. Equivalently, this "*blind*" reconfiguration scheme replaces high-complexity fault location techniques with CED schemes on various configuration attempts at run-time.

There are several ways to generate patterns for testing each configuration attempt. First, if the rollback recovery technique [Pradhan 96, Huang 00b] is used in the system for transient error recovery, one can use the same method to retry the input sequence that fails in the original computation. In the *rollback recovery* scheme, the system retries the faulty computations again starting from a certain time instant called *checkpoint*. The system state at each checkpoint is verified and stored. The input sequence since the previous checkpoint is also stored for the purpose of retrials. Although the test using rollback retry is closely integrated with the transient recovery scheme, such test sequence is not complete. There may be faults escaping such test sequence because they do not propagate to the output observed by the CED checker in alternative configurations. For example, the system may always feed a logic-0 signal on a stuck-at-0 wire in an alternative configuration attempt throughout the rollback recovery process. Although this stuck-at-0 wire is detected in the original configuration, it escapes the test sequence using rollback recovery. The effect of fault escapes during configuration attempts is further discussed in Sec. 4.1.

The second method is to use *pseudo-exhaustive BIST* (PE-BIST) patterns [McCluskey 81] to test each sub-circuit in the application running in the FPGA exhaustively. Such patterns provide very high fault coverage without relying on explicit fault models. Also, it does not require a large memory to store the test patterns because a minimal length PE-BIST pattern can be generated using a simple test pattern generator [McCluskey 82]. The test pattern generator can be implemented in the controller of the other FPGA in the dual-FPGA scheme.

The third method is to use a *linear feedback shift register* (LFSR) to apply *pseudo-random* test patterns to the circuit running in the FPGA [Abramovici 90]. Like PE-BIST, this approach does not require memory overhead to store test patterns, and the test patterns can be easily generated by the controller of the other FPGA in the dual-FPGA scheme. The fourth method is to use the *functional verification pattern* of the target application in the FPGA. Such pattern guarantees the correct functioning in the application, but it requires memory overhead to store the patterns.

Compared to the original circuit running in the FPGA, the major area overhead for this blind reconfiguration approach is the extra storage space and area reserved for precompiled configurations and the area for the CED scheme. However, both parts of the overhead are inherent either in the FPGA fault tolerance scheme or in the error detection scheme that is necessary for constructing a reconfigurable, dependable computing system. Therefore, this fault location approach does not cause major area overhead in a system that already has some CED scheme for data integrity and the precompiled configuration technique for fault tolerance.

In order to reduce the diagnostic time, one should choose a CED scheme with small error detection latency. For example, *duplex* CED that compares the results from duplicated modules in every cycle is feasible in minimizing the latency. On average, if there are k alternative configurations available for a k-column circuitry in the FPGA with 1-column fault tolerance in the overlapping precompiled configuration scheme, we need to try $k/2$ configurations assuming a single fault in the system. Therefore, for small circuits that are mapped within a few columns, this blind reconfiguration approach has potential in reducing system downtime caused by fault location.

Compared to the roving STAR approach in [Abramovici 99], this blind reconfiguration scheme guarantees data integrity and does not impose performance and availability degradation in fault-free operations. The control of the blind reconfiguration process is also simpler, and thus more feasible for FPGA-based dependable systems with autonomous recovery. However, there are some issues related to the blind reconfiguration scheme, and we discuss these issues in the following subsections. The issues include the effect of error detection capability in the CED scheme, the precision of fault location result, and the order of configuration attempts.

4.1 Effect of Error Detection Capability in CED

Because the blind reconfiguration approach uses the existing CED scheme to determine a successful configuration, data integrity is preserved after a successful configuration attempt. Consequently, even though some faults may escape a test sequence such as rollback retry patterns, data integrity is not compromised after reconfiguration providing faults are present only in the originally used part of the FPGA.

However, during the normal operation of the original configuration, permanent faults may occur in the reserved part of the FPGA and become *dormant* faults. In this case, when the next permanent fault occurs in the used part in the original configuration, such a single fault in the original mapped circuitry causes multiple faults in some alternative configuration attempts. Figure 4.1 illustrates this effect of dormant faults in the original configuration. The shaded area is unused, and the faulty area is marked with “x”. In this case, because of a dormant fault at (row 2, column 3), a single fault in the original mapped region at (row 1, column 2) causes multiple faults when the alternative configuration in Fig. 4.1(b) is attempted.

Note that this situation occurs when there are multiple permanent faults in the FPGA. To alleviate the effect of dormant faults, CED schemes with high coverage of multiple faults should be used. For example, *diverse duplex* systems with different implementations of the same logic function are good candidate because they provide better protection against multiple faults than other application-independent CED techniques [Mitra 00a].

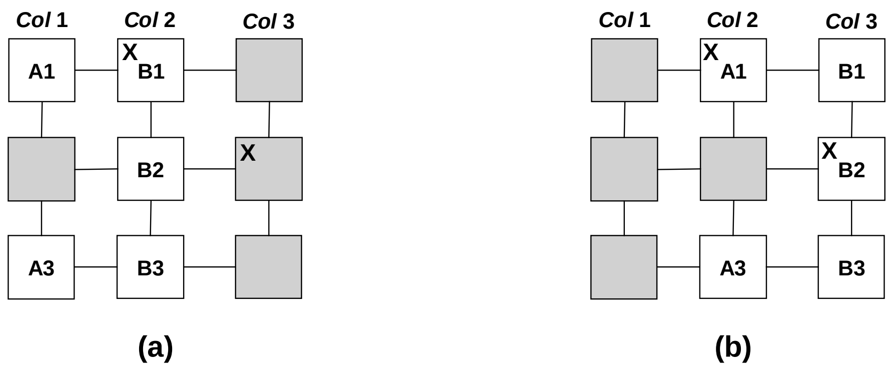

Figure 4.1: Example of the effect dormant faults. (a) Original configuration. (b) Alternative configuration when column 1 is intentionally unused.

4.2 Fault Location Precision

One important feature for the blind configuration attempts is that the result does not define a precise fault location in the FPGA. Instead, it finds a successful alternative configuration rapidly to resume the normal operation of the circuit in the FPGA. The actual fault may not occur in intentionally unused columns in a successful configuration attempt.

Figure 4.2 shows an example of the undefined fault location. Again, the shaded area is unused, and the faulty area is marked with "x". Because the configuration data is shifted in units of columns, the faulty block (row 2, column 2) is unused in the alternative configuration in Fig. 4.2(b) where column 1 is intentionally unused. This configuration attempt will be considered successful even though the actual fault does not occur in the intentionally unused column (column 1).

Because the actual fault location is not necessarily in intentionally unused columns in a successful configuration attempt, we should try all precompiled configurations except for the ones that fail in avoiding current faults when the next permanent fault occurs. For example, in Fig. 4.2, other configurations that use column 1 should not be completely excluded in the attempts when the next permanent fault occurs. This is to avoid the degradation of fault-tolerant capability (measured by the number of tolerable faulty columns in the FPGA) in the column-based precompiled configuration scheme.

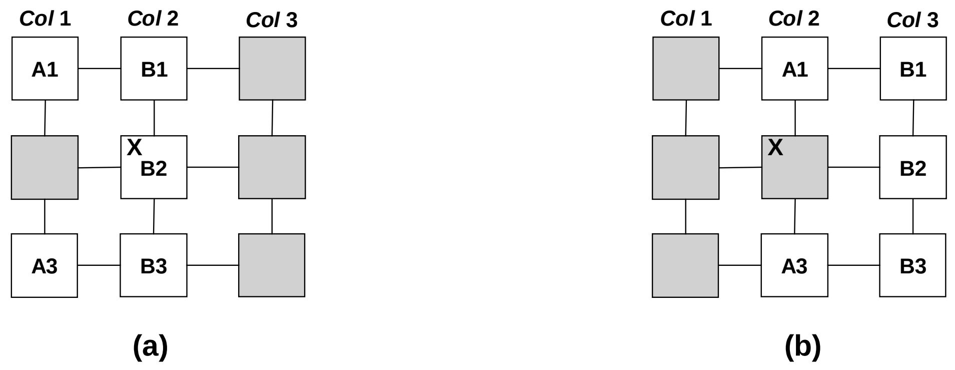

Figure 4.2: Example of undefined fault location. (a) Original configuration. (b) Alternative configuration when column 1 is intentionally unused.

4.3 Order of Configuration Attempts

In a CED scheme, the most crucial unit for guaranteeing data integrity is the CED checker. Therefore, the order of configuration attempts should be carefully arranged to ensure correct functioning of the CED checker.

Figure 4.3 shows an example of the impact of reconfiguration order on data integrity. Suppose the checker is mapped in the A3 block, and the faulty block is B3 in the original configuration. If the first configuration attempt avoids column 1, the checker is shifted to the faulty block and the checker output may be stuck at zero (error-free). In this way, the CED scheme fails to function correctly, and data integrity is not preserved in subsequent operations.

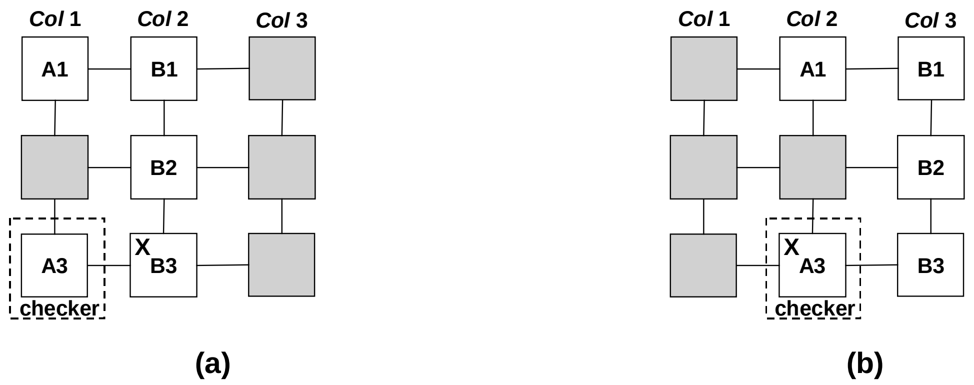

Figure 4.3: Effect of reconfiguration order. (a) Original configuration. (b) Alternative configuration when column 1 is intentionally unused.

To avoid corruption in the checker output, *self-checking checkers* should be used [McCluskey 90]. Such checkers guarantee data integrity for any single fault at the checker outputs. Another solution to guarantee the correct functioning in the checker is to try configurations without shifting the checker column (denoted as $Conf_{same\_checker}$) first. The system tries configurations that shift the checker column (denoted as $Conf_{shift\_checker}$) only when none of $Conf_{same\_checker}$ is successful.

Because the checker remains in the same position for $Conf_{same\_checker}$, the detection capability of the checker is identical to that of the normal operations. Also, the destination column for the checker in $Conf_{shift\_checker}$ (e.g., column 2 in Fig. 4.3) is intentionally unused in one of the configurations in $Conf_{same\_checker}$. If the destination column for the checker is the only faulty column, one of the configurations in $Conf_{same\_checker}$ can avoid the fault and becomes a successful attempt. The problem illustrated in Fig. 4.3 can thus be avoided by such an ordering of configuration attempts.

5. The Enhanced Fault Location Scheme: Minimal Configuration Attempts

5.1 Distributed CED Checkers

There are different levels of granularity at which CED can be performed. For reconfigurable systems, since it is possible to repair the system rather than replacing the faulty chip or the faulty board, it is reasonable to implement CED at a *distributed*, fine-grained level of *sub-circuits* in the system. For example, Fig. 5.1 shows the architecture of a duplex CED scheme with distributed checkers in each stage of a pipelined system. Other system-level architectures with such fine-grained CED have been described in [Mitra 00a].

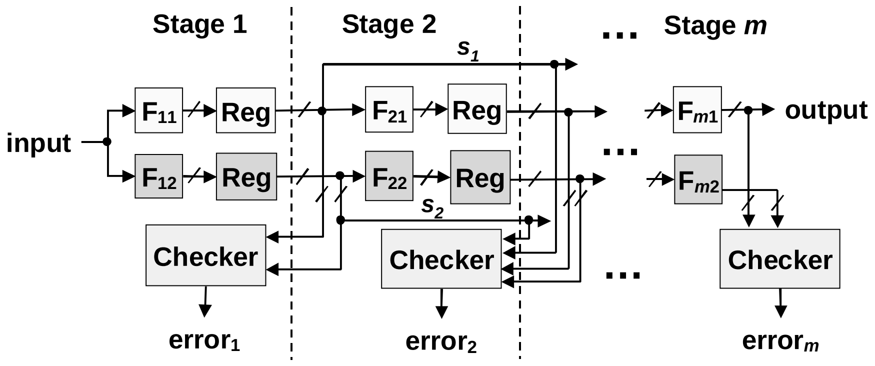

Figure 5.1: Duplex CED with distributed checkers in a pipelined system.

In Fig. 5.1, because pipeline registers are used to separate different stages in the system, errors signals from distributed CED checkers can localize faults within part of the system. If the CED checker that checks the *i*-th stage outputs does not signal an error in a certain cycle, correctness of computations in the *i*-th stage in this cycle can be ensured. In this way, we can reduce the number of suspect faulty columns in the FPGA and the number of configuration attempts.

Signals that propagate across more than two sub-circuits (e.g., both copies of signal s, s_1 and s_2, in Fig. 5.1) are also checked in intermediate sub-circuits if they are routed by PIPs in such intermediate sub-circuits. This is used for localizing the fault that is caused by an open in a long, global signal.

The outputs of the distributed checkers in each sub-circuit can be stored in flip-flops, which can be connected in a scan chain. When the system signals an error, the contents of the flip-flops storing the checker outputs can be scanned out for further processing by the controller in the other FPGA of the dual-FPGA architecture. The idea of using a scan chain to scan out the checker outputs is not new – it was used in the IBM mainframes to locate the faulty FRU (chip or board). The scanned data can be read out through a dedicated Scan Out pin or through the boundary scan port generally available in FPGA chips. If the number of checkers is moderate, dedicated I/O pins can be used to directly observe the checker outputs.

The controller stores information about the CLB columns that are occupied by each sub-circuit in the system. Such information is directly available from CAD tools, such as Xilinx Alliance software [Xilinx 01], at the design phase through the floorplan of each sub-circuit and checker in the FPGA. After the controller scans out the checker data, it can execute the following simple routine to find the set of suspect faulty columns:

```
Suspect = $\emptyset$
For each checker output do
    If the checker produces an error signal
        Suspect = Suspect $\cup$ {Columns occupied by the sub-circuit that is checked
            by the checker}
    Endif
Endfor
```

Note that we use the union operation for finding the suspect set because a fault in a global signal that connects multiple sub-circuits causes error reports in multiple checkers. In this case, every sub-circuit that use or propagate this signal can be suspect. If the suspect set contains only one CLB column, the configuration to be loaded is the one that does not use the suspect column. If the suspect set contains multiple CLB columns, the configurations to be loaded are restricted to those that do not use at least one of the suspect CLB columns.

Suppose that there are *m* sub-circuits in the system, each of which has a localized, distributed CED checker. For this distributed checker scheme, the worst-case number of configuration attempts for avoiding single fault that does not occur on wires across multiple sub-circuits is $max(k_1, k_2, \ldots, k_m)$, where k_i is the number of CLB columns occupied by the *i*-th sub-circuit. The worst-case number of configuration attempts for avoiding single fault that occurs on a wire across multiple sub-circuits is the summation of k_i's in every sub-circuit along the path of the wire.

Note that since the faulty column suspects are specified by distributed checkers, our technique can also be integrated with the roving STAR approach in [Abramovici 99] in order to find the precise fault location. The only difference is that, instead of roving the STAR across the entire FPGA, we only need to rove the STAR across the suspect columns. The resulting number of reconfigurations required in the roving STAR approach is thus reduced when integrating with the distributed CED checkers in our approach.

Generally, for fine-grained partitioning of sub-circuits, the number of configuration attempts is smaller than systems with coarse-grained sub-circuits. This is because each sub-circuit occupies fewer columns in the FPGA in systems partitioned with finer granularity. However, there is a possible tradeoff with area overhead because of the increasing number of distributed checkers in the system because of fine-grained partitioning. We discuss this issue using an example in FPGAs in Sec. 5.3.

5.2 Modified Column-based Precompiled Configuration Scheme

By floorplanning the design, each sub-circuit and its corresponding CED checker can be confined within a certain region in the FPGA. In this way, we can minimize the number of configuration attempts by using a modified column-based precompiled configuration scheme, which is illustrated in Fig. 5.2.

In Fig. 5.2, instead of shifting part of the configuration in units of CLB columns, alternative configurations in the modified scheme are created by shifting logic mappings in units of sub-circuits. Each alternative configuration avoids the entire mapped region of a sub-circuit in the original configuration by such sub-circuit-based shifting. Because the suspect faulty columns are formed based on the distributed checkers in each sub-circuit, the number of configurations to be loaded is minimized by such sub-circuit-based shifting strategy.

For single fault that does not occur on wires across multiple sub-circuits, the suspect faulty columns are confined within one sub-circuit. In this case, only one reconfiguration is necessary for the entire FPGA fault location and recovery process. For single fault that occurs on wires across *n* sub-circuits, the worst-case number of configuration attempts is *n*.

To guarantee single-fault tolerance, the modified column-based precompiled configuration scheme needs to reserve $max(k_1, k_2, \ldots, k_m)$ CLB columns in the original configuration. Compared to the scheme in Sec. 5.1, this enhanced approach minimizes the number of configuration attempts with the price of extra CLB columns that are reserved as backup.

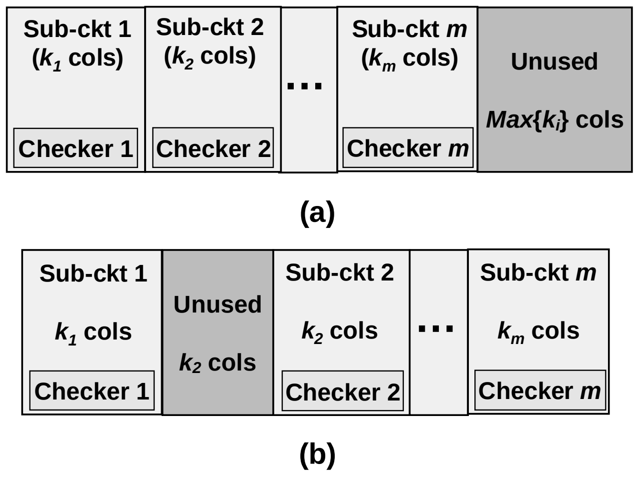

Figure 5.2: The modified column-based precompiled configuration. (a) Original configuration. (b) Alternative configuration when checker 2 reports error.

Because alternative configurations are created by shifting multiple columns, the same coding scheme in the original column-based precompiled configuration approach described in [Huang 01b] can be used for compressing configuration data and achieving significant storage reduction. For an *m*-sub-circuit system, the total number of alternative configurations required for tolerating single fault is *m*. Compared to the overlapping column-based precompiled configuration scheme where $(k_1+k_2+\ldots+k_m)$ alternative configurations are required for guaranteeing single-fault tolerance, the modified scheme needs fewer configurations and thus a smaller configuration storage overhead.

Note that when a sub-circuit occupies a large number of columns, an alternative configuration that avoids the whole region of such sub-circuit may not be found because of the routing constraints and the total number of columns available in the FPGA. Therefore, the modified column-based precompiled configuration approach is more feasible when the system can be partitioned into smaller sub-circuits.

5.3. Case Study: LZ Compressor in FPGAs

In this section, we present the estimation of area overhead due to the fine-grained partitioning and distributed checkers in the enhanced scheme. The area overhead is estimated using a case study, the Lempel-Ziv (LZ) compression application [Ziv 77], in FPGAs. The LZ encoder circuitry is shown in Fig. 5.3, which is identical to the *systolic-array* approach in [Jung 98] and [Huang 00a].

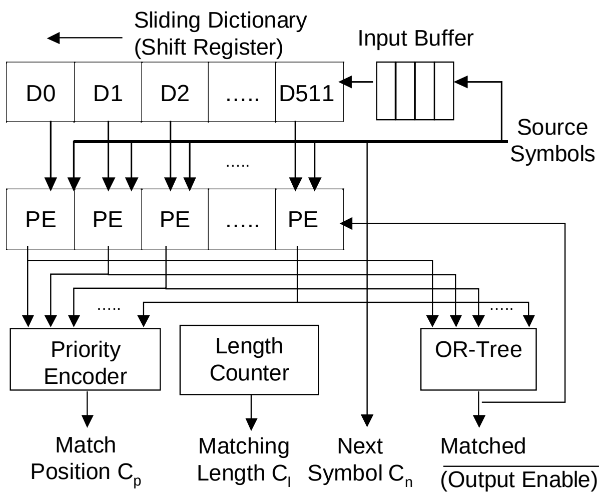

Figure 5.3: A systolic-array LZ encoder.

[Huang 00a] proposed an area-efficient *inverse comparison* CED scheme for the LZ encoder in FPGAs. The inverse comparison CED scheme detects the error by checking if the encoded output can be decoded to match the source data correctly. However, because the check is only performed at the output codewords, it is difficult to partition the LZ encoder and to insert distributed checkers in fine granularity if the inverse comparison CED is used. Therefore, in this case study where the entire LZ encoder is the target system, we use the duplex CED scheme for estimating the impact of area overhead due to the fine-grained partitioning and distributed checkers. Using a duplex CED scheme, 100% single fault coverage can be assured.

In Fig. 5.3, the LZ encoder consists of a systolic array of 512 shift registers with corresponding *Processing Elements* (PEs). Each shift register is 8-bit wide, and each PE is basically a comparator that matches source symbols with data in shift registers. The priority encoder takes the match results from the PE array and encodes the match position in the array. The OR-tree takes the match results from the PE array and determines if a match is found. A length counter is used for counting the matching length, and there are input and output buffers and controllers for coordinating input and output sequences. A detailed circuit can be found in [Jung 98] and [Huang 00a].

Fine-grained partitioning of the LZ encoder can be realized by grouping multiple shift registers, PEs, and their corresponding encoding circuitry in the priority encoder and OR-tree as sub-circuits. Length counters, input controllers, and output controllers can be grouped as another sub-circuit. The resulting partitioned LZ encoder architecture with *m* sub-circuits is shown in Figure 5.4. A duplex CED scheme with an output comparator is implemented in each sub-circuit, and outputs in each sub-circuit are latched in order to localize the fault at the sub-circuit level.

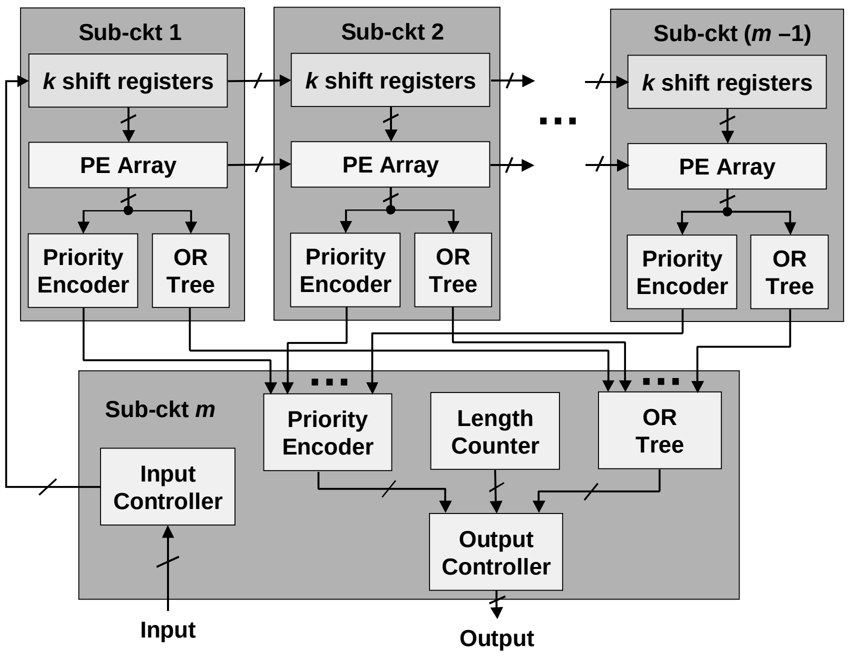

Figure 5.4: Partitioned LZ encoder.

The parameter *m*, the number of sub-circuits in the system, in Fig. 5.4 is determined by considering the following trade-offs. First, each column should be occupied by no more than one sub-circuit, and we can choose a smaller *m* to reflect this factor. If a column is divided into multiple sub-circuits, multiple CED checkers are used to check different parts of the column. Such fine-grained sub-circuit partitioning is overkill in the column-based precompiled configuration scheme because the diagnostic resolution requirement is at the level of columns. Excessive checkers in a column may cause large area overhead without improving fault location latency.

Second, *m* should be big enough so that sub-circuits 1 to (*m* – 1) (partitions of the systolic array) are not significantly larger than the *m*-th sub-circuit (global control logic), which is relatively fixed in size. Large difference in sub-circuit sizes generally requires more columns to be reserved as backup in the modified column-based precompiled configuration scheme discussed in Sec. 5.2. This is because the size of the reserved area is determined by the size of the largest sub-circuit. It is thus more desirable to partition the system into sub-circuits with similar sizes.

Our designs are mapped in the Xilinx Virtex XCV1000 FPGA [Xilinx 01], which has a CLB array of 64 rows and 96 columns. The area and clock frequency results are reported by Xilinx Alliance 3.1i place-and-route tool. The global control block with duplex CED (the *m*-th sub-circuit in Fig. 5.4) is mapped into five columns. In order to balance sub-circuit sizes, the parameter *m* is chosen between 9 or 17 (i.e., 8 or 16 sub-circuits in the systolic array and 1 sub-circuit for global control), which results in a size of 10 columns or 5 columns, respectively, for each sub-circuit with duplex CED (sub-circuit 1 to (*m* – 1) in Fig. 5.4) in the systolic array.

Table 5.1 shows the resulting area and clock frequency for simplex, duplex without partitioning, duplex with nine sub-circuits, and duplex with 17 sub-circuits. Note that although the area overhead in terms of CLB utilization increases with the number of sub-circuits, the number of columns used does not change significantly in different duplex schemes. This is because the place-and-route software generally results in scattered empty CLBs in order to facilitate routing. Therefore, the area overhead due to distributed checkers and fine-grained partitioning is not very significant compared to a duplex scheme without partitioning. Degradation of the maximum clock rate because of partitioning is within 10% compared to the duplex scheme without partitioning.

Table 5.1: Comparison of area overhead and clock rate.

Scheme	CLB utilization (slices)	CLB columns used	Max. clock rate (MHz)
1. Simplex	4863	41	33.2
2. Duplex without partition	9743	84	32.7
3. Duplex with 9 sub-circuits	9794	85	31.5
4. Duplex with 17 sub-circuits	9962	85	30.0

Table 5.2 summarizes the partitioning results for duplex schemes with 9 and 17 sub-circuits. Because there are global signals connecting the controller sub-circuit and all the other sub-circuits in the systolic array, the worst-case suspect set spans the entire used region in the FPGA. However, such global signals represent only about 0.2% of all signals in the system in both schemes, and only faults in such signals can produce a suspect faulty set of size greater than two sub-circuits. Therefore, with a very high probability, the FPGA fault location and recovery process can be completed in only one or two reconfigurations using the modified column-based precompiled configuration scheme described in Sec. 5.2.

Table 5.2: Comparison of two partitioning schemes.

Scheme	No. of checkers	Max. no. of columns per sub-ckt	Max. no. of columns in the suspect set	Percentage of signals across more than 2 sub-ckts
Duplex, 9 sub-ckts	9	10	85	0.19%
Duplex, 17 sub-ckts	17	5	85	0.22%

For the duplex scheme with 9 sub-circuits, 10 additional columns are required as backup for guaranteeing 1-column fault tolerance using the modified column-based precompiled configuration scheme. For the duplex scheme with 17 sub-circuits, only 5 additional columns are required to achieve the same fault tolerance capability. Since both schemes are mapped into the same number of columns, the duplex scheme with 17 sub-circuits is preferred if the modified column-based precompiled configuration scheme is used.

6. Conclusion

We presented a new technique, which integrates CED schemes with the column-based precompiled configuration approach used for FPGA fault tolerance, for solving run-time FPGA fault location and recovery problem rapidly. Using distributed CED checkers in each floorplanned sub-circuit of a system, faulty column suspects in the FPGA can be obtained, and the number of configuration attempts for fault location and recovery can be reduced. With a modified column-based precompiled configuration scheme that shifts the configuration in units of floorplanned sub-circuits, the number of configuration attempts can be further minimized.

As a case study, we implemented our technique in the LZ compression application with duplex schemes in Xilinx XVC1000 FPGAs. For duplex CED schemes, extra area overhead and variations of clock frequencies due to partitioning and distributed checkers are small. When the modified column-based precompiled configuration scheme is used, a duplex scheme with 17 sub-circuits is preferred and can complete the FPGA fault location and recovery process in one or two reconfigurations with a very high probability.

Acknowledgements

The authors would like to thank Dr. Nirmal Saxena, Dr. Santiago Fernandez-Gomez, Chao-Wen Tseng, Chien-Mo Li, and Shu-Yi Yu for their valuable feedback and suggestions. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-97-C-0024.

Reference

[Abramovici 90] Abramovici, M., M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Revised printing, IEEE Press, 1990.

[Abramovici 99] Abramovici, M., C. Stroud, C. Hamilton, et. al., "Using Roving STARs for Online Testing and Diagnosis of FPGAs in Fault-Tolerant Applications," *Proc. International Test Conference*, pp. 973-982, 1999.

[Carmichael 99] Carmichael, C., E. Fuller, P. Blain, and M. Caffrey, "SEU Mitigation Techniques for Virtex FPGAs in Space Applications," *MAPLD '99*, Sept. 1999.

[Das 99] Das, D., and N.A. Touba, "A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems," *Proc. of IEEE International Conference on VLSI Design*, pp. 266-269, 1999.

[Emmert 98] Emmert, J., and D. Bhatia, "Incremental Routing in FPGAs," *Proc. of 11th Annual IEEE International ASIC Conference*, pp. 217-221, 1998.

[Emmert 00] Emmert, J., C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 165-174, 2000.

[Hanchek 98] Hanchek, F., and S. Dutt, "Methods for Tolerating Cell and Interconnect Faults in FPGAs," *IEEE Trans. on Computers*, Vol. 47, No. 1, pp. 15-32, 1998.

[Hauck 98] Hauck, S., "The Roles of FPGA's in Reprogrammable Systems," *Proceedings of the IEEE*, Vol. 86, No. 4, pp. 615-638, 1998.

[Huang 00a] Huang, W.-J., N. Saxena, and E. J. McCluskey, "A Reliable LZ Data Compressor on Reconfigurable Coprocessors," *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 249-258, 2000.

[Huang 00b] Huang, W.-J., and E. J. McCluskey, "Transient Errors and Rollback Recovery in LZ Compression," *Proc. 2000 Pacific Rim International Symposium on Dependable Computing*, pp. 128-135, 2000.

[Huang 01a] Huang, W.-J., and E. J. McCluskey, “A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations,” *Proc. Ninth ACM International Symposium on Field-Programmable Gate Arrays*, pp. 183-192, 2001.

[Huang 01b] Huang, W.-J., and E. J. McCluskey, “Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance,” *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001, to appear.

[Jordan 93] Jordan, C., and W.P. Marnane, “Incoming Inspection of FPGAs,” *Euro. Test Conf.*, pp. 371-377, 1993.

[Lach 98] Lach, J., W. H. Mangione-Smith, and M. Potkonjak, “Efficiently Supporting Fault-Tolerance in FPGAs”, *Proc. ACM International Symposium on Field-Programmable Gate Arrays*, pp. 105-115, 1998.

[Lakamraju 00] Lakamraju, V., and R. Tessier, “Tolerating Operational Faults in Cluster-Based FPGAs,” *Proc. ACM International Symposium on Field Programmable Gate Arrays,* pp. 187-194, 2000.

[Lombardi 96] Lombardi, F., D. Ashen, X. Chen, and W.K. Huang, “Diagnosing Programmable Interconnect Systems for FPGAs,” *Int’l Symp. on FPGAs*, pp. 100-106, 1996.

[Lucent 01] Lucent Technologies, *http://www.lucent.com*, 2001.

[Mahapatra 99] Mahapatra, N. R., and S. Dutt, “Efficient Network-Flow Based Techniques for Dynamic Fault Reconfiguration in FPGAs”, *FTCS’99*, pp. 122-129, 1999.

[McCluskey 81] McCluskey, E.J., and S. Bozorgui-Nesbat, “Design for Autonomous Test,” *IEEE Trans. Comp.*, pp. 866-875, Nov. 1981.

[McCluskey 82] McCluskey, E.J., “Verification Testing,” *Proc. Design Automation Conferenc*e, pp. 495-500, 1982.

[McCluskey 90] McCluskey, E. J., “Design techniques for Testable Embedded Error Checkers,” *IEEE Compute*r, Vol. 23, No. 7, pp. 84-88, July 1990.

[Mitra 98] Mitra, S., P. P. Shirvani, and E.J. McCluskey, “Fault Location in FPGA-Based Reconfigurable Systems,” *IEEE Intl. High Level Design Validation and Test Workshop,* 1998.

[Mitra 00a] Mitra, S. and E.J. McCluskey, “Which Concurrent Error Detection Scheme to Choose?,” *Proc. International Test Conference,* pp. 985-994, 2000.

[Mitra 00b] Mitra, S., W.-J. Huang, N. R. Saxena, S.-Y. Yu, and E. J. McCluskey, “Dependable Adaptive Computing Systems: The Stanford CRC ROAR Project,” *Fast Abstracts, 2000 Pacific Rim International Symposium on Dependable Computing,* pp. 15-16, 2000.

[Pradhan 96] Pradhan, D. K., *Fault-Tolerant Computer System Design,* Prentice Hall, 1996.

[Renovell 98] Renovell, M., J.M. Portal, J. Figueras, and Y. Zorian, “Testing the Interconnect of RAM-Based FPGAs,” *IEEE Design & Test of Computers,* pp. 45-50, Jan.-Mar. 1998.

[Saxena 00] Saxena, N. R., S. Fernandez-Gomez, W.-J. Huang, S. Mitra, S.-Y Yu, and E. J. McCluskey, "Dependable Computing and On-Line Testing in Adaptive and Configurable Systems," *IEEE Design and Test*, Vol.17, No. 1, pp. 29-41, 2000.

[Stroud 97] Stroud, C., E. Lee, and M. Abramovici, "BIST-Based Diagnostics of FPGA Logic Blocks," *Proc. International Test Conference*, pp. 539-547, 1997.

[Stroud 98] Stroud, C., S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-In Self-Test of FPGA Interconnect," *Proc. International Test Conference*, pp. 404-411, 1998.

[Touba 97] Touba, N. A. and E. J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. CAD*, Vol. 16, pp. 783-789, July 1997.

[Trimberger 98] Trimberger, S., "Scheduling Designs into a Time-Multiplexed FPGA," *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 153-160, 1998.

[Xilinx 01] Xilinx Inc., *http://www.xilinx.com*, 2001.

[Zeng 99] Zeng, C., N. R. Saxena and E. J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection," *Proc. Intl. Test Conf.*, pp. 672-680, 1999.

[Ziv 77] Ziv, J., and A. Lempel, "A Universal Algorithm for Sequential data Compression," *IEEE Trans. Information Theory*, Vol. IT-23, No. 3, pp. 337-343, 1977.