Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance

Wei-Je Huang and Edward J. McCluskey
Center for Reliable Computing
Department of Electrical Engineering
Stanford University, Stanford, CA 94305
{weije, ejm}@crc.stanford.edu

Abstract

The abundance of configurable logic elements and routing resources in recent Field-Programmable Gate Arrays (FPGAs) provides a cost-effective method for tolerating permanent faults in the system. Once a permanent fault occurs, the FPGA can be reconfigured by replacing the faulty part with previously unused resources in the same hardware. In this paper, we present two column-based precompiled configuration techniques for tolerating permanent faults in FPGA-based systems. By compiling alternative configuration versions in the design phase, these approaches ensure fast reconfiguration, and thus a tremendous increase in system availability. In addition, intentional similarities are created among different configuration versions so that the storage overhead due to precompiled configurations is reduced by orders of magnitude through differential coding and run-length coding. Experimental and analytical results show that our approaches achieve significant dependability improvement with small configuration storage overhead.

1. Introduction

With the rapid progress of process technology and device architecture, the capacity and performance of Field-Programmable Gate Arrays (FPGAs) have been boosted by orders of magnitude in recent years. Because of the abundance of configurable logic and routing resources, current FPGAs became widely used in various applications. Typical applications that use FPGAs extensively include not only logic emulation systems, but also commercial networking and storage equipment that may be used in mission-critical environment. Therefore, it is crucial to enhance the fault tolerant capability and improve the dependability of FPGA-based systems.

To tolerate permanent faults in a system, hardware redundancy is the most commonly used approach [Siewiorek 92][Pradhan 96]. Traditionally, hardware redundancy is realized in a *coarse-grained* level. All functional modules in the system are replicated such that permanent faults in part of the system can be tolerated.

However, the coarse-grained hardware redundancy is expensive in terms of area overhead. This is especially noticeable in FPGA-based systems because one can find alternative mappings of a functional module in FPGAs to avoid certain parts of the device. Therefore, instead of the coarse-grained hardware redundancy, a more cost-effective method is to reconfigure the FPGA such that the faulty parts are replaced with previously unused resources in the same device. Equivalently, this method realizes the hardware redundancy in a *fine-grained* level. In this way, the system can still operate in the presence of faults, and dependability is improved with very little hardware redundancy [Saxena 98].

Extensive research has been done in the past in order to achieve permanent fault recovery in FPGAs through reconfiguration. This includes fast re-mapping and rerouting techniques for dynamic run-time generation of alternative configurations [Emmert 97, 98][Hanchek 98][Dutt 99][Mahapatra 99], and the tile-based pre-compiled configuration approach that creates alternative configurations in the design phase [Lach 98, 99].

Generally, the precompiled configuration approach minimizes the system downtime because alternative configuration versions are pre-generated. Thus, the re-mapping and rerouting of user application circuitry is not necessary once the fault location is diagnosed. However, this approach also results in a significant storage overhead for all possible configurations in order to achieve a high coverage of faults. With the increasing size of FPGA configuration data, the extra cost due to the configuration storage overhead may outweigh the cost reduction through the fine-grained hardware redundancy. In this situation, such reconfiguration methods become unattractive for fault tolerance, especially in deep-space and unmanned applications.

Another drawback of previous approaches is the assumption that high-precision fault location techniques are available prior to reconfiguration. However, although there are many fault location techniques in the literature [Stroud 97, 98][Mitra 98][Das 99], these techniques require high computation complexity to diagnose a fault in the level of a Configurable Logic Block (CLB) or a Programmable Interconnect Point (PIP). Consequently, these techniques are difficult to implement at run-time and increase the overall downtime of the system.

In this paper, we present the concept of a column-based precompiled configuration technique for tolerating permanent faults in FPGAs in a cost-effective way. Our approach mitigates the configuration storage overhead and fault location problems of previous techniques. Different configurations are generated by shifting some or all of the CLB columns in the initial configuration along one direction in order to create similarities among configurations. Such intentional similarities result in smaller storage overhead through differential coding and data compression techniques. In addition, high-precision fault location operations prior to reconfiguration can be replaced by gross fault location or different precompiled configuration attempts.

The organization of this paper is as follows. In Sec. 2, we describe the FPGA architectural model used in this paper. In Sec. 3, we briefly discuss previous work for constructing dependable FPGA-based systems. In Sec. 4, we introduce two column-based precompiled configuration techniques, the *overlapping* and the *non-overlapping* schemes, for permanent fault recovery in FPGAs. A data compression technique for storage reduction and a method for replacing high-precision fault location techniques are also presented. In Sec. 5, we compare the dependability improvement of the two column-based precompiled configuration techniques. In Sec. 6, configuration storage overhead improvements for some of the MCNC benchmarks are presented. Section 7 concludes the paper.

2. FPGA Architecture

Figure 1 shows the model of the programmable logic core of an FPGA used in this paper. In this model, the programmable logic core of an FPGA consists of an array of three basic elements: *Configurable Logic Blocks* (CLBs), *Connection Boxes* (CBs), and *Switch Boxes* (SBs).

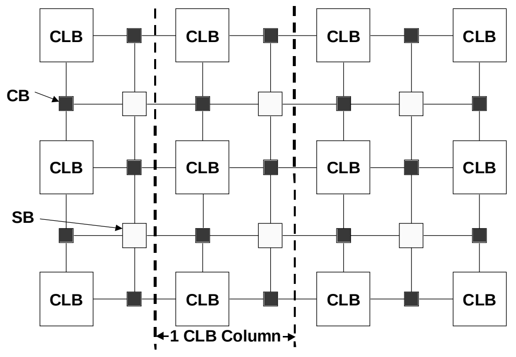

Figure 1: Architecture of the programmable logic core in FPGAs.

In SRAM-based FPGAs, a CLB contains several SRAM lookup tables (LUTs) to store user-defined logic functions. A CLB also contains flip-flops, multiplexers, and dedicated circuitry to optimize the performance of user applications. CLBs are connected through horizontal and vertical wiring channels between two neighboring rows or columns. To enhance the connectivity, there are various kinds of wires with different lengths for connecting CLBs that are different blocks apart. 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].

There are two types of routing devices, CBs and SBs, to direct the signal flows among CLBs and wiring channels. CBs serve as a local bridge between CLBs and the adjacent wiring channels. SBs are switch matrices that connect horizontal and vertical wiring channels. In SRAM-based FPGAs, the state of connections in these routing devices is 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. Figure 2 shows the configuration data frame architecture in this type of FPGA [Xilinx 01].

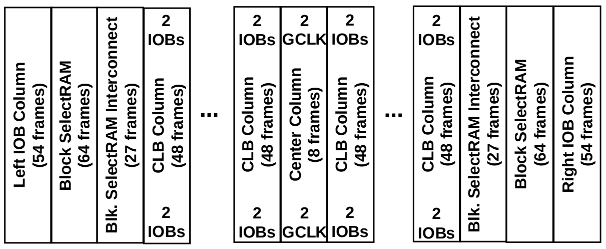
Left IOB Column (54 frames)
Block SelectRAM (64 frames)
Blk. SelectRAM Interconnect (27 frames)
2 IOBs
CLB Column (48 frames)
2 IOBs
...
2 IOBs
CLB Column (48 frames)
2 IOBs
2 GCLK
Center Column (8 frames)
2 GCLK
2 IOBs
CLB Column (48 frames)
2 IOBs
...
2 IOBs
CLB Column (48 frames)
2 IOBs
Blk. SelectRAM Interconnect (27 frames)
Block SelectRAM (64 frames)
Right IOB Column (54 frames)

Figure 2: Configuration frame architecture in Xilinx Virtex-series FPGAs.

3. Related Work

Previous research on dependable FPGA-based systems can be classified into the following categories: the dependable system architecture, Concurrent Error Detection (CED) schemes for the applications mapped in FPGAs, fault location and diagnosis techniques, transient error recovery, and permanent fault recovery.

Figure 3 shows a dual-FPGA architecture for the dependable adaptive computing systems proposed in Stanford CRC ROAR project [Mitra 00b]. In this architecture, each FPGA is configured to run certain applications with some CED schemes. The controller on each FPGA monitors the error signal from the CED schemes on the other FPGA and performs the fault location and recovery for the other FPGA when necessary.

Various CED techniques can be found in [Saxena 00][Huang 00a][Mitra 00a]. These approaches verify the correctness of the system by additional redundant computations either in the space or time domain. The error detection latency for these techniques is small in order to report the occurrence of errors rapidly and reduce the performance degradation.

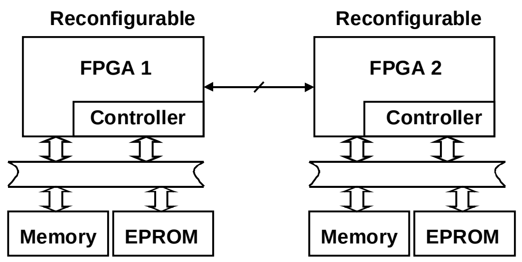

Figure 3: Dependable dual-FPGA architecture.

Generally, the first response of a system to the detection of errors is the transient error recovery operation because transient errors occur more frequently than permanent faults in the hardware. Traditional system-level approaches, such as rollback or roll-forward schemes [Siewiorek 92][Pradhan 96][Huang 00b], can be used to recover from transients that do not affect the FPGA configuration memory. Also, for FPGAs with the partial reconfiguration feature, configuration readback and writeback operations can be performed simultaneously with system-level recovery approaches to recover from transients that alter the data stored in the FPGA configuration memory [Carmichael 99][Huang 01].

When transient error recovery schemes fail to recover the chip from an error in several attempts, fault location and permanent fault recovery schemes are initiated to repair the system from permanent faults. For fault location in FPGAs, the *Built-In Self-Test* (BIST) technique can be used. In the BIST approach, a diagnostic circuit including test pattern generators and response analyzers is configured in part of the FPGA to test another part pseudo-exhaustively and vice-versa. The diagnosis can be application-dependent [Mitra 98][Das 99] or application-independent [Stroud 97, 98], according to whether the circuit under tests is configured in the same way as the original application circuitry. In both cases, however, a considerable number of configurations are required to test all the hardware resources thoroughly and locate the fault. For high precision and resolution, these techniques generally require high computation complexity and long diagnostic latency.

After the faulty part of an FPGA is located, permanent fault recovery schemes that reconfigure the FPGA to replace the faulty part with originally unused resources are applied to repair the system. Several approaches were proposed in the literature in order to generate alternative configuration rapidly at run-time. Emmert and Bhatia proposed a minimax grid matching technique to re-map the functional units that are originally placed in faulty CLBs [Emmert 97]. They also proposed an incremental routing technique to reroute the corresponding signals [Emmert 98]. For *cluster-based* FPGAs that group multiple LUT and flip-flop pairs into a single cluster to take the advantage of design locality, Lakamraju and Tessier proposed a localized swapping technique to effectively tolerate intra-cluster faults [Lakamraju 00]. Also, Dutt et. al. proposed node covering techniques that reserve spare resources in order to facilitate the search for the optimal replacement for faulty parts [Dutt 99][Hanchek 98] [Mahapatra 99].

To further reduce the system downtime for permanent fault recovery, Lach et. al. [Lach 98, 99] proposed a tile-based precompiled configuration approach that move the generation of alternative configurations to the design phase. The mapped circuitry is partitioned into tiles, and alternative configurations for each tile are pre-generated and stored in the system. The drawback of the precompiled configuration approach is the significant storage overhead in the system for all possible alternative configurations. Although the tile-partitioning approach in [Lach 98] reduces the storage requirement for alternative configurations to some extent, such storage overhead is still several times of the original configuration size.

In this paper, we assume that the dependable dual-FPGA architecture in Fig. 3 is used. CED and transient error recovery schemes for applications in each FPGA are also applied prior to permanent fault recovery. Our goal is to develop a permanent fault recovery scheme for FPGAs that minimizes the system downtime and has small configuration storage overhead. To minimize the system downtime, we use the precompiled configuration approach for fast reconfiguration. To reduce the configuration storage overhead, we propose two column-based design methodologies and a coding scheme for effective compression of configuration data. In addition, unlike previous permanent fault recovery techniques, our approach does not require high-precision, high-complexity fault location techniques prior to reconfiguration.

4. Column-Based Precompiled Configuration Techniques

In order to reduce the storage overhead in the system due to precompiled configurations, data compression techniques that reduce redundancy in the information can be applied. Most of the data compression techniques reduce data size by encoding highly correlated information with shorter codewords. Consequently, for a good compression ratio, the configuration data should be highly correlated.

For the data within one configuration version, it is not guaranteed to obtain highly correlated configuration bit-stream in arbitrary applications. However, for the data among different configuration versions, high correlation can be created intentionally by proper selections of re-mapping, rerouting, and configuration data partitioning methods when different configuration versions are constructed during the design phase.

As mentioned in Sec. 2, the target FPGA architecture in this paper has identical circuitry, routing resources, and configuration architecture in every CLB column. Therefore, our precompiled configuration approach is designed to create intentional similarities based on CLB columns in different configuration versions. According to the part of FPGA used in the original configuration and alternative configuration versions, we can classify our column-based precompiled configuration technique into two schemes: the *overlapping* scheme and the *non-overlapping* scheme.

4.1 The Overlapping Scheme

The key concept of the overlapping precompiled configuration scheme can be illustrated by an example shown in Fig. 4. In Fig. 4(a), suppose that the original fault-free configuration, or the *base configuration*, is mapped in four consecutive CLB columns (column 1 to column 4). The *column-based functional modules* that are mapped in each of the four columns are function *A*, *B*, *C*, and *D*, respectively. The entire circuitry is thus defined by the four column-based functional modules and the interconnects among these modules.

Let us consider the case for tolerating faults within any single CLB column. To this objective, we reserve one column outside of the mapped area (column 5) in the base configuration as backup resources for alternative configurations. All CLBs and switch matrices in the reserved column are unused. Also, the configuration of each column-based functional module in the mapped area is stored separately as a basis to construct alternative configurations.

In this case, four alternative configurations are required in order to guarantee 1-column fault tolerance in the FPGA. In each alternative configuration, one of the mapped columns (column 1 to 4) in the base configuration is intentionally unused. All functional modules originally mapped in CLB columns with smaller column indices than the intentionally unused column remain in the same places in the alternative configuration. The other functional modules are shifted rightwards by one column in the alternative configuration to avoid the intentionally unused column.

For example, Figure 4(b) shows one of the alternative configurations, where column 3 is intentionally unused. Functional modules mapped in column 1 and column 2 in the base configuration (function *A* and *B*) remain in the same places. Functional modules mapped in column 3 and column 4 in the base configuration (function *C* and *D*) are shifted rightwards to column 4 and column 5, respectively, to avoid using column 3.

In the overlapping precompiled configuration scheme, the above re-mapping procedure in each alternative configuration creates several corresponding column sets. A *corresponding column set* is defined as a set of CLB columns in which certain functions are mapped in different configuration versions. For example, column 3 in Fig. 4(a) and column 4 in Fig. 4(b) belong to the same corresponding column set in which function *C* is mapped, and there are four corresponding column sets in this case. *Corresponding columns* in different configurations are defined as CLB columns in the same corresponding column set that holds a certain column-based functional module.

In Fig. 4(b), the re-mapping of the functional modules forms two *mapped regions* that are separated by the intentionally unused column. The left mapped region contains column 1 and 2 (function *A* and *B*), and the right mapped region contains column 4 and 5 (function *C* and *D*). In this way, only *inter-region* signals that connect functional units in different mapped regions need to be rerouted due to the shifted functional modules. *Intra-region* signals that connect functional units within the same mapped region, however, can be routed in the same way as their counterparts in the corresponding base configuration column. This is because every CLB column has the same programmable logic and routing resources. Equivalently, the routing information of intra-region signals can be obtained directly by shifting the states of switches in corresponding base configuration columns using the same method described in the re-mapping procedure for column-based functional modules.

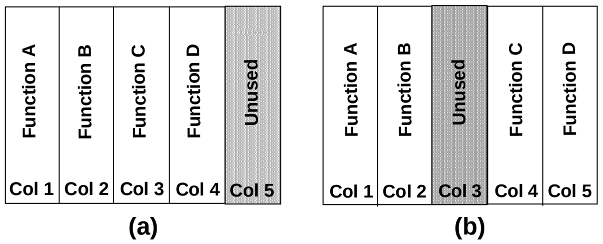

Figure 4: The overlapping precompiled configuration. (a) Base configuration. (b) Alternative configuration when column 3 is intentionally unused.

The overlapping precompiled configuration scheme with 1-column fault tolerance can be generalized to a scheme for tolerating multiple faulty CLB columns as follows. First, if the desired *number of tolerable faulty columns* is *m*, which indicates that the scheme can tolerate faults within any *m* CLB columns, we need to reserve *m* unused columns as backup resources in addition to the mapped area in the base configuration. In this case, if *the number of column-based functional modules in the base configuration* is *k* (i.e., *k* CLB columns are used to map the original circuit in the base configuration), the overlapping scheme with *m*-column fault tolerance requires (*k*+*m*) CLB columns in the FPGA. The (*k*+*m*) CLB columns are indexed from left to right.

Second, for the overlapping scheme with *m*-column tolerance, *m* out of (*k*+*m*) CLB columns are intentionally unused in each configuration version. The total number of configurations required is thus $C(k+m, m) = (k+m)! / (m!k!)$, with one configuration being the base configuration. Therefore, one needs to construct $[C(k+m, m) - 1]$ alternative configurations in order to achieve *m*-column tolerance in a *k*-column circuit.

Next, in each alternative configuration, all column-based functional modules (in unit of a CLB column) that are originally mapped in CLB columns with smaller column indices than the least-indexed intentionally unused column remain in the same places. The other column-based functional modules are shifted rightwards by one column. If a column-based functional module is shifted to another intentionally unused column in this alternative configuration, it is shifted one more column rightwards along with all the subsequent functional modules. In this way, the (k+m) columns can be partitioned into several mapped regions that are separated by m intentionally unused columns. Switch states that route intra-region signals are shifted in the same way as the k column-based functional modules.

Finally, inter-region signals are rerouted to complete the alternative configuration. In order to avoid using the switch resources in intentionally unused columns, wires that connect CLBs of multiple blocks apart can be used in rerouting inter-region signals. For example, in Xilinx Virtex-series FPGAs [Xilinx 01], hex lines can be used to route signals across different mapped regions without using the switch resources in intentionally unused columns.

One drawback of this scheme is that the maximum number of horizontal routes used in each column is limited by the number of horizontal multiple-block wires available to reroute inter-region signals. When horizontal multiple-block wires are not available, another solution for rerouting signals across intentionally unused columns is to use single-block wires and the corresponding switches that are not used in the base configuration. In both cases, at least half of the routing resources for horizontal wires are reserved for rerouting.

Nevertheless, the utilization of routing resources for vertical wires is not limited by this rerouting scheme because all vertical connections are intra-region and need not be rerouted. Therefore, to accommodate the horizontal routing constraint, the base configuration should be constructed such that most of the signals flow in the vertical direction.

The advantage of this scheme is that different configurations are very similar because each alternative configuration is created by shifting part of the base configuration in units of columns. More similarity in configuration data leads to a good compression ratio, and thus small storage overhead, when data compression techniques are applied. This property will be further examined in Sec. 4.3 when we discuss the configuration data compression technique

4.2 The Non-overlapping Scheme

If the target circuitry is small enough to fit within half of the FPGA, a simple way to construct alternative configurations is to shift the entire mapped circuitry to originally unused regions. This is the *non-overlapping* precompiled configuration scheme, where there is no overlap between the base configuration and alternative configurations.

In order to tolerate up to m faulty columns in the FPGA, the total number of alternative non-overlapping configurations required in addition to the base configuration is m. In this case, the base configuration has to be mapped within 1/(m+1) of the entire FPGA columns. Therefore, one drawback is the limitation in the size of the mapped region in the base configuration.

Still, the non-overlapping scheme can be feasible in situations where various applications are implemented in an FPGA in a time-multiplexed manner to minimize cost. In such cases, the FPGA is chosen to accommodate the largest circuit size among all target applications. Because of variations in circuit sizes, there may be small application circuits in which the size constraint in the non-overlapping scheme is satisfied.

Compared to the overlapping scheme, the non-overlapping scheme has smaller storage overhead for alternative configurations. This is caused by two factors. First, there are only m alternative configuration required for m-column fault tolerance, which is fewer than the [$C(k+m, m) - 1$] configurations in the overlapping scheme.

Second, the non-overlapping scheme results in more similarity among different configuration versions because the relative positions among the mapped column-based functional modules are preserved. In this way, the only difference between the corresponding columns in different configurations is from the connections to primary inputs and outputs of the device. More similarity between configurations result in a better compression ratio, which will be examined in Sec. 4.3.

4.3 Configuration Data Compression

In the proposed precompiled configuration schemes, CLB columns of different configurations in the same corresponding column set have the greatest similarity. The only difference between such corresponding columns in different configurations is the inter-region reroutes and the wires to the primary I/O’s of the device. All LUT entries and the switch states that control the vertical connections and the intra-region horizontal connections, on the other hand, are identical in the corresponding columns in different configurations

In this case, the bit-wise *difference vectors* in the corresponding columns between the base configuration and alternative configurations contain strings of long, consecutive 0’s and scattered 1’s (may have short run-lengths). Therefore, *run-length coding*, which encodes a string of 0’s with its length, is expected to be very effective in reducing the storage of these difference vectors.

In our proposed scheme, we use *Golomb codes* [Golomb 66] to encode the difference vectors between the corresponding columns in the base configuration and alternative configurations. Also, note that it is not necessary to store the mapping relationship of corresponding columns. This is because such information can easily be derived given that the intentionally unused columns are already specified in each alternative configuration.

Given the configuration data, C_b, of the corresponding base configuration column, the encoding algorithm for the configuration data of a CLB column, C_a, in an alternative configuration is described as follows:

(1) Find the bit-wise *difference vector* $C_d = C_a \oplus C_b$. The difference vector consists of *strings of* S_z*'s*, which are defined as z leading 0's (run-length $z \geq 0$) followed by a 1.

(2) A *group size* $g = 2^n$ (n is a positive integer) is selected in encoding each string S_z in C_d.

(3) Each string S_z, with z leading 0's, is encoded as a codeword C_w. Each codeword is the concatenation of a *group code* and a *tail code*. The group code is defined as $\lfloor z / g \rfloor$ leading 1's followed by a 0, and the tail code is the n-bit binary representation of (z mod g). An example with group size $g = 4$ is shown in Table 1.

Table 1: Example of Golomb code with group size = 4.

z	Source String S_z	Group Code	Tail Code
0	1	0	00
1	01		01
2	001		10
3	0001		11
4	00001	10	00
5	000001		01
6	0000001		10
7	00000001		11
8	000000001	110	00
9	0000000001		01
10	00000000001		10
11	000000000001		11

Golomb code with group size g transforms a $(z+1)$-bit source data string into a codeword of $(\lfloor z / g \rfloor + 1 + \log_2 g)$ bits. When the run-length of 0's in the source data string is significantly larger than the group size g, the compression ratio approximates to $1/g$.

Clearly, the compression ratio improves with increasing run-length of 0's in the source data string. Because of intentional similarities created among different configurations, the proposed column-based schemes are expected to obtain long run-lengths in difference vectors, and thus, a good compression ratio. Also, using run-length coding, the non-overlapping scheme achieves more data compression than the overlapping scheme due to greater similarity in the corresponding CLB columns.

In addition to a good compression ratio, another factor that makes the proposed differential and run-length coding suitable for encoding configuration data is the simple decoding process. Given the base configuration column, C_b, whose index can be derived from the intentionally unused columns specified for the alternative configuration, the decoding process of a codeword, C_w, is described as follows:

(1) Count the length, L_1, of the leading 1's in C_w before encoutering the first zero.

(2) Multiply L_1 by the group size g. Since $g = 2^n$, the multiplication is equivalent to shifting L_1 towards the most-significant-bit (MSB) by n bits.

(3) Add the following n-bit tail code in the C_w to the result in (2) to obtain the run-length z of 0's in the original difference vector. Equivalently, this is to append the n-bit tail code to the end of L_1 obtained in (2). The difference vector, C_d, is then constructed by appending a 1 to z leading 0's.

(4) Reconstruct the alternative configuration column, C_a, by $C_a = C_b \oplus C_d$. Equivalently, we can flip the $(z+1)$-th bit in C_b to obtain C_a.

The implementation of the proposed decoding process in hardware requires only a counter to compute the run-length and XOR circuitry to reconstruct alternative configuration columns from difference vectors. This simple decoding property is critical in minimizing both the area overhead and the system downtime due to the fault recovery process by FPGA reconfiguration.

Note that corresponding columns can be configured exactly in the same way for different configuration versions. For example, in the case of Fig. 4, the CLB column that holds the function *A* can have the same settings for the base configuration and any alternative configuration where no inter-region reroute is required for this function.

Therefore, for each alternative configuration, we store the pointers to the encoded difference vectors instead of the actual data of such vectors. In this way, the actual data of each encoded difference vector are stored only once, and different configuration versions can share the same difference vector without redundant storage. In addition, for each alternative configuration, we store the column indices of intentionally unused columns that are required to locate the corresponding base configuration columns.

4.4 Fault Location

In the precompiled configuration approach, if the fault location is specified, reconfiguration can be initiated promptly by downloading an appropriate alternative configuration stored with the system. Previous approaches require fine-grained resolution, usually in the level of one CLB, in the fault location techniques. For the proposed schemes, on the contrary, this high-resolution requirement for fault location techniques is avoided.

For the proposed precompiled configuration schemes, the requirement for fault location resolution is to specify a faulty column instead of a faulty CLB. The coarse resolution requirement generally reduces the complexity of fault location techniques and thus makes such techniques more feasible.

If fault location techniques are not available, all possible configurations stored with the system can be tried alternately until an appropriate configuration that operates successfully is set up in the system. CED schemes for the application circuitry, as discussed in Sec. 3, are used to determine if the reconfiguration attempt is successful. Equivalently, this "*blind*" reconfiguration scheme replaces high-complexity fault location techniques with CED techniques on various configuration attempts at run-time.

Generally, CED schemes have shorter error detection latency than the diagnostic latency in fault location techniques. Although CED schemes are unable to diagnose the fault location, they are effective in detecting the occurrence of errors online. Therefore, this approach has great potential to reduce the system downtime caused by fault location operations prior to reconfiguration and is useful for mission-critical and deadline-critical applications.

5. Dependability Improvement

In this section, we analyze the dependability improvement of the proposed schemes using the parameter of *Mean Time to Failure* (MTTF). MTTF is defined as the expected time of the first failure in the system, given successful startup at time zero [Siewiorek 92].

Without any permanent fault recovery scheme, a system could fail to function properly once a permanent fault occurs in part of the circuitry. Therefore, to simplify the analysis, we define the *failure in an FPGA-based system without permanent fault recovery schemes* as the occurrence of a permanent fault in any part of the mapped columns in the FPGA.

In contrast, we define the *failure with the proposed precompiled configuration schemes* as the situation when no precompiled configuration version is available to avoid the faulty part of the FPGA. There are two assumptions in this definition. First, we assume that the target FPGA remains configurable throughout the period of interest. This assumption allows us to focus on the dependability improvement in the programmable logic core of the FPGA using the proposed scheme.

Second, we assume that the system downtime due to reconfiguration is negligible compared to the duration between the occurrence of permanent faults in the FPGA. Practically, this is a reasonable assumption because of the low occurrence rate for permanent faults and the fast reconfiguration of current-generation FPGAs.

In addition, the following assumptions are made for the analysis:

(1) A constant *failure rate* λ is associated with each CLB column in the FPGA. This is the probability of the occurrence of a permanent fault in a CLB column.

(2) k CLB columns are used in the base configuration.

(3) The occurrence of faults in each CLB column is independent.

The *reliability* of a functional module is defined as the probability that the module operates successfully from time zero to time t, given that it commences successfully at time zero [Siewiorek 92]. For a constant failure rate in each CLB column, the reliability of a CLB column, $R_{col}(t)$, follows the exponential failure law, which states that

$$R_{col}(t) = e^{-\lambda t}.$$

Because there are k CLB columns in the base configuration with independent occurrence of faults, the reliability of the base configuration without any fault recovery scheme, $R_{base}(t)$, is

$$R_{base}(t) = \left(R_{col}(t)\right)^k = e^{-\lambda k t}.$$

From [Siewiorek 92], the MTTF of a system can be calculated by the integral of the reliability function:

$$MTTF = \int_0^\infty R(t)dt.$$

Therefore, the MTTF of the base configuration without any fault recovery scheme, $MTTF_{base}$, is

$$MTTF_{base} = \int_0^\infty e^{-\lambda k t}dt = \frac{1}{\lambda k}.$$

For the overlapping scheme with m-column fault tolerance, m CLB columns are reserved to tolerate up to m faulty columns in the base configuration. Therefore, the reliability of the overlapping scheme, $R_{ov}(t)$, becomes

$$\begin{aligned}R_{ov}(t) &= prob\begin{pmatrix} \text{no more than } m \text{ faulty columns} \\ \text{out of } (k+m) \text{ columns}\end{pmatrix} \\ &= \sum_{n=0}^{m}\left[C(k+m,n)\cdot e^{-\lambda t(k+m-n)}\cdot(1-e^{-\lambda t})^n\right] \\ &= \sum_{n=0}^{m}\left[C(k+m,n)e^{-\lambda t(k+m-n)}\sum_{j=0}^{n}C(n,j)(-e^{-\lambda t})^j\right] \\ &= \sum_{n=0}^{m}\sum_{j=0}^{n}\left[C(k+m,n)C(n,j)(-1)^j e^{-\lambda(k+m-n+j)t}\right].\end{aligned}$$

The MTTF of the overlapping scheme, $MTTF_{ov}$, is thus

$$\begin{aligned}MTTF_{ov} &= \int_0^\infty R_{ov}(t)dt \\ &= \sum_{n=0}^{m}\sum_{j=0}^{n}\left[\frac{(-1)^j C(k+m,n)C(n,j)}{\lambda(k+m-n+j)}\right] \\ &= \frac{\sum_{n=0}^{m}\sum_{j=0}^{n}\left[(-1)^j\frac{kC(k+m,n)C(n,j)}{(k+m-n+j)}\right]}{\lambda k}.\end{aligned}$$

For the non-overlapping scheme with m-column tolerance capability, m disjoint k-column regions are reserved in addition to the original k-column base configuration. Therefore, the reliability of the non-overlapping scheme, $R_{no}(t)$; becomes

$$R_{no}(t) = prob\begin{pmatrix} at\ least\ one\ out\ of\ the\ (m+1) \\ regions\ is\ good\ up\ to\ time\ t \end{pmatrix}$$

$$= 1 - prob\left(all\ regions\ are\ faulty\ at\ time\ t\right)$$

$$= 1 - \left[1 - \left(e^{-\lambda t}\right)^k\right]^{m+1}$$

$$= 1 - \sum_{n=0}^{m+1}\left[C(m+1,n)\left(-e^{-\lambda kt}\right)^n\right]$$

$$= \sum_{n=1}^{m+1}\left[\left(-1\right)^{n-1}C(m+1,n)e^{-\lambda nkt}\right].$$

The MTTF of the non-overlapping scheme, $MTTF_{no}$, is thus

$$MTTF_{no} = \int_0^\infty R_{no}(t)dt$$

$$= \frac{\sum_{n=1}^{m+1}\left[\left(-1\right)^{n-1}\frac{C(m+1,n)}{n}\right]}{\lambda k}.$$

Figure 5 shows the *normalized MTTF* for the two proposed schemes with respect to different numbers of tolerable faulty columns, *m*. The MTTF's for both schemes are normalized to $MTTF_{base}$ (i.e., multiplied by λk) to indicate the dependability improvement relative to the original circuitry without any fault recovery technique. Note that only the overlapping scheme is dependent on the circuit size, *k*. In Fig. 5, we choose three different circuit sizes (*k* = 10, 20, and 50) for the overlapping scheme. As a comparison, current-generation FPGAs, such as the Xilinx Virtex-E family, have choices ranging from 24 to 156 CLB columns in a chip.

In Fig. 5, it is clear that both schemes can achieve significant MTTF improvement. With the same number of tolerable faulty columns, the overlapping scheme has a better MTTF than the non-overlapping scheme. Intuitively, this can be explained in two reasons. First, in order to guarantee *m*-column fault tolerance, the non-overlapping scheme requires more CLB columns (totally $(m+1)k$ columns) to implement than the overlapping scheme (totally $(k+m)$ columns). A greater number of CLB columns used in the non-overlapping scheme results in a larger area and becomes more susceptible to faults.

Second, given the same number of columns, N_{col}, in the FPGA for implementing a *k*-column circuit using both schemes, the overlapping scheme is guaranteed to tolerate $(N_{col} - k)$ faulty columns, whereas the non-overlapping scheme is guaranteed to tolerate only $(\lfloor N_{col} / k \rfloor - 1)$ faulty columns. The difference is more noticeable when the circuit size, *k*, is large.

Figure 6 shows the normalized MTTF increment for each additional column of tolerance in both schemes. The *normalized MTTF increment* represents the incremental MTTF improvement resulted from adding the fault-tolerance capability by one more column in both schemes. The result follows the law of diminishing return, which indicates that the maximal gain in MTTF improvement is obtained when the system is changed from a no-recovery scheme to a scheme with 1-column tolerance capability.

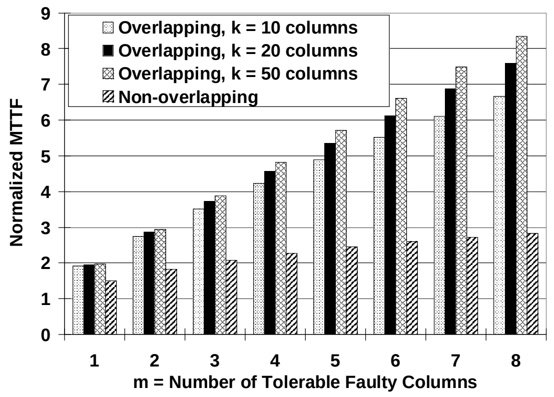

Figure 5: Normalized MTTF for the proposed schemes.

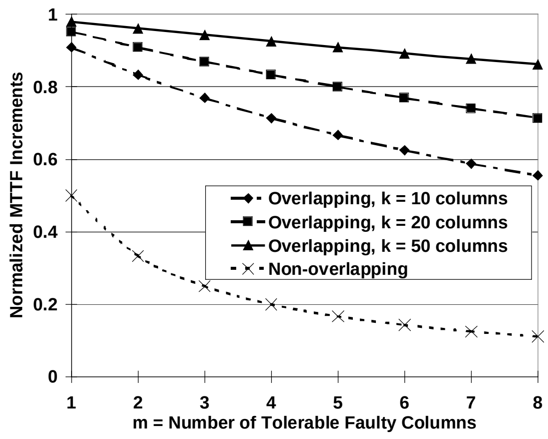

Figure 6: Normalized MTTF increments.

6. Configuration Storage Overhead and Performance Impact

To demonstrate the storage overhead improvement for alternative configurations, we applied the proposed schemes with 1-column tolerance capability (*m* = 1) to a subset of MCNC benchmark circuits. Because the benchmark circuits are relatively small compared to the capacity of current-generation FPGAs, we used the smallest FPGA in Xilinx Virtex-E series, XCV50E [Xilinx 01], in our experiments. This type of FPGA has a 16x24 CLB array in each device.

The number of CLB columns (parameter *k*) used in the base configuration for each benchmark circuit is shown in Table 2. In each base configuration circuit, we constrained the mapped area and placed the I/O's within a certain columns in the left part of the chip. In this way, we can reduce the number of alternative configurations required in the overlapping scheme and minimize the routings in the horizontal direction.

Table 2: Size of benchmark circuits.

Circuit	Number of CLB columns used in the base configuration (*k*)
c499	11
duke2	10
planet1	6
sand1	5

Because the number of tolerable faulty columns in the experiments is one, *k* alternative configurations for the overlapping scheme and one alternative configuration for the non-overlapping scheme are required, respectively. Therefore, without data compression, the system has *k*-times storage overhead in the overlapping scheme and 100% storage overhead in the non-overlapping scheme in addition to the storage of the base configuration.

To generate alternative configurations, we manipulated the Xilinx Design Language (XDL) files [Xilinx 01] of base configurations according to the proposed methods. XDL files describe the physical mapping of functional units and the routing of nets in FPGAs in text format. The resulting XDL files for both base configurations and alternative configurations can be translated into configuration bit-streams, which are processed according to the configuration architecture in [Xilinx 00] in order to extract the configuration data of corresponding columns. The extracted configuration data for each column is then encoded as difference vectors and compressed using Golomb codes with different group sizes.

Figure 7 and 8 shows the resulting storage overhead for the benchmark circuits due to the encoded alternative configurations in both precompiled configuration schemes, respectively. The storage overhead is calculated relative to the part of configuration data required for the used columns in the base configuration, instead of the overall configuration bit-stream for the device. This is to avoid the over-optimistic results because of the small size of the benchmark circuits relative to the capacity of the FPGA.

From Fig. 7 and Fig. 8, a group size of 128 in the Golomb code minimizes the storage overhead of alternative configuration versions in both precompiled configuration schemes. For the overlapping scheme, the resulting minimum storage overhead of alternative configurations is in the range of 15-35% for the benchmark circuits. For the non-overlapping scheme, the minimum storage overhead of alternative configurations is around 2-6% only. As discussed in Sec. 4.2, the small configuration storage overhead in the non-overlapping scheme comes at the price of more CLB columns for implementing the same circuit with a given number of tolerable faulty columns.

Compared to the multiple-time configuration storage overhead without compression in previous approaches, the proposed schemes achieve 1-2 orders of magnitude improvement in storage requirement for alternative configurations.

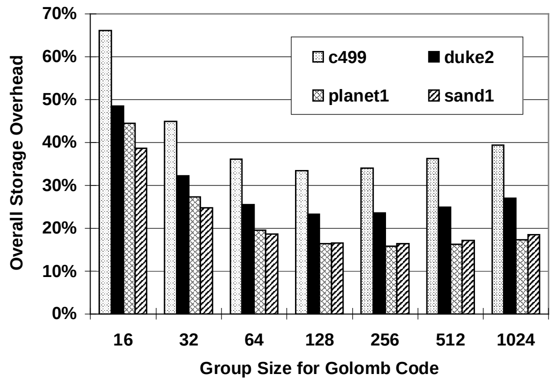

Figure 7: Storage overhead for the overlapping precompiled configuration scheme.

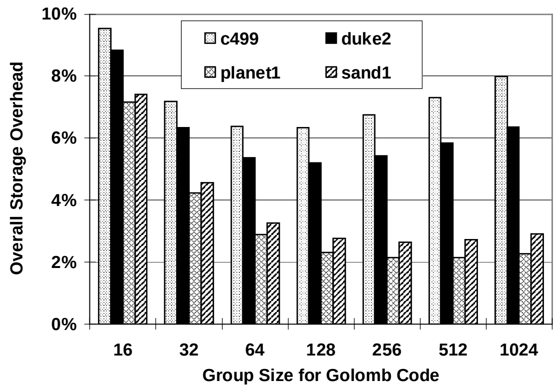

Figure 8: Storage overhead for the non-overlapping precompiled configuration scheme.

In order to evaluate the performance impact in alternative configurations due to shifting and rerouting, we measured the maximum combinational path delay in each configuration for different circuits. The results are reported by the timing analyzer tool in Xilinx Alliance 3.1i package. Compared to the base configuration, the worst-case critical path delay overhead in the alternative configurations for both schemes range from 11% to 18%. Also, for some alternative configurations where the critical path is not changed due to shifting and rerouting, there is no performance degradation after reconfiguration.

7. Conclusions

In this paper, we presented two column-based precompiled configuration techniques, the overlapping scheme and the non-overlapping scheme. By creating alternative configurations that avoid certain parts of the original mapped area in the FPGA during the design phase, the precompiled configuration approach improves the dependability of FPGA-based systems significantly and achieves fast reconfiguration for reducing the system downtime.

As a comparison of the proposed schemes, the overlapping scheme has a better MTTF improvement, while the non-overlapping scheme achieves smaller storage overhead for alternative configurations but needs more CLB columns for implementation. Because similarity among alternative configurations is intentionally created, the storage overhead for alternative configurations in both schemes is reduced by orders of magnitude using differential and run-length coding. Also, high-precision, high-complexity fault location operations prior to reconfiguration can be replaced with concurrent error detection schemes on alternative configuration attempts. Both schemes result in graceful performance degradation after reconfiguration.

Acknowledgements

The authors would like to thank Dr. Nirmal Saxena, Dr. Santiago Fernandez-Gomez, Dr. Subhasish Mitra, Chao-Wen Tseng, Chien-Mo Li, Philip Shirvani, 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.

References

[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 Int'l Conf. on VLSI Design*, pp. 266-269, 1999.

[Dutt 99] Dutt, S., V. Shanmugavel, and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in Field Programmable Gate Arrays," *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design*, pp. 173-176, 1999.

[Emmert 97] Emmert, J. M., and D. Bhatia, "Partial Reconfiguration of FPGA Mapped Designs with Applications to Fault Tolerance and Yield Enhancement," *Proc. Int'l Workshop of Field-Programmable Logic*, pp. 141-150, 1997.

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

[Golomb 66] Golomb, S. W., "Run-length Encoding," *IEEE Trans. on Information Theory*, Vol. IT-12, pp. 399-401, 1966.

[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.

[Huang 00a] Huang, W.-J., N. Saxena, and E. J. McCluskey, "A Reliable LZ Data Compressor on Reconfigurable Coprocessors," *Proc. IEEE Symp. 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 Int'l Symp. on Dependable Computing*, pp. 128-135, 2000.

[Huang 01] Huang, W.-J., and E. J. McCluskey, "A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations," *Proc. Ninth ACM Int'l Symp. on Field-Programmable Gate Arrays*, pp. 183-192, 2001.

[Lach 98] Lach, J., W. H. Mangione-Smith, and M. Potkonjak, "Efficiently Supporting Fault-Tolerance in FPGAs", *Proc. ACM Int'l Symp. on Field-Programmable Gate Arrays*, pp. 105-115, 1998.

[Lach 99] Lach, J., W. H. Mangione-Smith, and M. Potkonjak, "Algorithms for Efficient Runtime Faulty Recovery on Diverse FPGA Architectures", *DFT'99*, pp. 386-394, 1999.

[Lakamraju 00] Lakamraju, V., and R. Tessier, "Tolerating Operational Faults in Cluster-Based FPGAs," *Proc. ACM Int'l Symp. on Field Programmable Gate Arrays*, pp. 187-194, 2000.

[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.

[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. Int'l Test Conference*, 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 Int'l Symp. on Dependable Computing*, pp. 15-16, 2000.

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

[Saxena 98] Saxena, N. R., and E. J. McCluskey, "Dependable Adaptive Computing Systems," *IEEE Systems, Man, and Cybernetics Conf.*, pp. 2172-2177, Oct. 11-14, 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.

[Siewiorek 92] Siewiorek, D. P., and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, 2nd Edition, Digital Press, 1992.

[Stroud 97] Stroud, C., E. Lee, and M. Abramovici, "BIST-Based Diagnostics of FPGA Logic Blocks," *Proc. Int'l 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. Int'l Test Conference*, pp. 404-411, 1998.

[Xilinx 00] Xilinx Application Note, "XAPP151: Virtex Configuration Architecture Advanced Users' Guide," *http://www.xilinx.com/xapp/xapp151.pdf*, 2000.

[Xilinx 01] Xilinx Inc., *http://www.xilinx.com*, 2000.